Couchbase

WHITEPAPER

Pourquoi des bases de données NoSQL ?

Pourquoi les entreprises à succès s'appuient sur les applications de bases de données NoSQL

Contents

Qu'est-ce que le NoSQL ?

NoSQL est un type moderne de système de gestion data qui stocke les informations dans des documents JSON au lieu des colonnes et des lignes utilisées par les bases de données relationnelles. Il fournit des données aux applications d'une manière qui rend le développement plus facile et plus robuste.

Par conséquent, les bases de données NoSQL sont conçues dès le départ pour être flexibles, évolutives et capables de répondre rapidement aux exigences de gestion des données des entreprises modernes basées sur Internet.

Ce document présente les défis modernes auxquels sont confrontées les bases de données NoSQL, et montre quand & pourquoi choisir NoSQL plutôt que SQL.

L'expérience client pousse les entreprises à adopter les solutions NoSQL

La révolution NoSQL a été motivée par la demande croissante d'interactions engageantes avec les clients et la nécessité de développer un avantage concurrentiel, en particulier avec les applications web en temps réel.

L'expérience client est le principal facteur de différenciation concurrentielle et les entreprises s'efforcent de répondre à ces nouvelles attentes en proposant des services à la demande, en temps réel, résilients et réactifs.

Les entreprises d'aujourd'hui interagissent numériquement — non seulement avec leurs clients, mais aussi avec leurs employés, leurs partenaires, leurs fournisseurs et même leurs produits — à une échelle sans précédent.

Ces applications alimentées par le réseau, la périphérie et le web sont au coeur de la révolution NoSQL : l'Internet des objets (IoT), les réseaux sociaux, l'analyse big data, le Cloud interne ou externe d'une entreprise, l'informatique mobile/edge, etc.

Pour réunir tous ces nouveaux systèmes, il faut plus de souplesse, d'évolutivité, et de meilleures performances, qui permettront de regrouper plusieurs types de systèmes en une seule plate-forme.

Relationnel ou NoSQL : Quelle est la différence ?

Les bases de données relationnelles sont nées à l'époque des mainframes et des applications commerciales de back-office — bien avant Internet, le Cloud, le big data, le mobile et l'économie numérique distribuée d'aujourd'hui, basée sur la 5G. En fait, la première implémentation commerciale a été lancée par Oracle en 1979. Ces bases de données ont été conçues pour fonctionner sur un seul serveur — plus c'est gros, mieux c'est. La seule façon d'augmenter la capacité de ces bases de données était de mettre à niveau les serveurs — processeurs, mémoire et stockage — afin de les faire évoluer au rythme de la loi de Moore.

Les bases de données NoSQL sont apparues à la suite de la croissance exponentielle de l'internet et de l'essor des applications web. La recherche "Google Bigtable" a été publiée en 2006, et la recherche Amazon Dynamo en 2007. Des moteurs distribués efficaces de type clé-valeur (KV) ont été essentiels à cette évolution et ont propulsé la technologie bien plus loin.

De nouvelles bases de données ont été conçues pour répondre à la nouvelle génération d'exigences des entreprises, que des sociétés comme Couchbase ont poussées encore plus loin pour répondre aux besoins à venir — la nécessité de **se développer avec agilité** et de **fonctionner à n'importe quelle échelle**.

L'agilité consistait à fournir des schémas flexibles, des API utiles, des requêtes SQL robustes, des analyses de texte, etc. Tandis que l'évolutivité permettait d'augmenter les volumes de données sans sacrifier les performances et la stabilité.

Soutien aux développeurs SQL et NoSQL

Les systèmes relationnels traditionnels se contentent de gérer des données tabulaires et de les restituer sous forme de lignes et de colonnes. Les bases de données NoSQL peuvent également faire cela, mais les applications NoSQL modernes ne forcent pas les développeurs d'applications à utiliser un schéma statique qui doit être retravaillé à chaque changement. Au contraire, les bases de données NoSQL offrent aux développeurs la flexibilité dont ils ont besoin pour les aider à exceller dans leur travail.

Les systèmes NoSQL contiennent des données JSON hiérarchiques mais les renvoient à l'application sous la forme de structures de données JSON complètes ou partielles, de résultats de recherche en texte intégral, de lignes de requêtes SQL tabulaires, d'objets clés, voire de systèmes d'analyse de données volumineuses.

Cette convergence des technologies simplifie l'architecture de l'information des entreprises et aide les développeurs à fournir des applications plus efficacement sans avoir à apprendre une douzaine de plateformes différentes.

Mise à l'échelle au-delà des bases de données SQL

Pour fonctionner à l'échelle, la nouvelle approche des systèmes NoSQL nécessitait une informatique en cluster avec des interconnexions de noeuds efficaces pour maintenir la synchronisation et le flux des données à grande vitesse.

Par exemple, on attend désormais des responsables techniques qu'ils mettent en place une infrastructure de gestion des données d'entreprise présentant les caractéristiques suivantes :

- Prise en charge d'un grand nombre d'utilisateurs simultanés (des dizaines de milliers, voire des millions)
- Offrir des expériences très réactives à une base d'utilisateurs répartis dans le monde entier
- Disponibilité continue — pas de temps d'arrêt
- Traiter des données semi-structurées et non structurées
- S'adapter rapidement à l'évolution des besoins grâce à des mises à jour fréquentes et à de nouvelles fonctionnalités

Les bases de données relationnelles basées sur SQL sont incapables de répondre à ces nouvelles exigences, alors que les bases de données NoSQL le peuvent.

Voici quelques exemples d'entreprises du Global 2000 qui déploient du NoSQL pour des applications critiques et qui ont fait l'objet de reportages récents :

- **Tesco,** leader européen de la distribution. Le premier détaillant européen déploie du NoSQL pour son commerce électronique, son catalogue de produits et d'autres applications
- **Ryanair,** une des compagnies aériennes les plus fréquentées au monde, utilise du NoSQL pour alimenter son application mobile destinée à plus de 3 millions d'utilisateurs
- **Marriott** déploie du NoSQL pour son système de réservation qui rapporte 38 milliards de dollars par an
- **Viber** traite plus de 15 milliards d'événements par jour pour ses plus de 1 milliard de clients
- **GE** déploie du NoSQL pour sa plateforme Predix afin d'aider à gérer l'internet industriel

Les tendances d'aujourd'hui — les défis de demain

Les objectifs actuels en matière d'expérience client dépendent plus que jamais d'une intégration technique étroitement alignée, mais celle-ci doit s'adapter parfaitement aux tendances de l'économie numérique à venir, sous peine de devenir obsolète.

- Des plates-formes consolidées qui fonctionnent ensemble efficacement
- Des architectures de système simplifiées et faciles à gérer
- Une pipeline de données efficace pour les applications web en temps réel et une faible latence
- Agir comme une couche de service qui pousse les données aussi près que possible du client

Quatre avantages des bases de données NoSQL dans l'économie numérique actuelle

Les tendances plus larges de l'économie numérique des entreprises ont introduit de nouveaux défis qui poussent les objectifs ci-dessus encore plus loin. Des clients ambitieux ayant de grandes idées sur la manière d'utiliser les données ont déclenché une nouvelle série d'exigences technologiques pour les DSI et les responsables techniques.

Voici quatre tendances qui mettent en évidence les avantages des bases de données NoSQL pour relever les défis de l'économie numérique.

Tendances de l'économie numérique	Exigences
La clientèle continue de se tourner vers les services en ligne	• Possibilité de prendre en charge des milliers, voire des millions d'utilisateurs • Répondre aux exigences de l'utilisateur avec des performances élevées et constantes • Maintenir une disponibilité 24 heures sur 24, 7 jours sur 7
Internet se connecte à tout	• Prise en charge de nombreuses applications différentes avec des structures de données différentes • Veiller à ce que le logiciel soit "toujours opérationnel", sans aucune excuse pour les temps d'arrêt • Prise en charge de flux continus de données provenant du web en temps réel
Le Big Data prend de l'ampleur	• Stockage des données semi-structurées et non structurées générées par les clients • Stockage de différents types de données provenant de différentes sources dans la même infrastructure, voire dans le même cluster • Stocker les données générées par des milliers ou des millions de clients et d'objets
Les applications se déplacent vers le Cloud	• Évolution à la demande pour prendre en charge davantage de clients et stocker davantage de données • Exploitation d'applications entièrement gérées à l'échelle mondiale pour soutenir des clients dans le monde entier • Minimiser les coûts d'infrastructure et accélérer la mise sur le marché

Les exigences ci-dessus sont vastes, et mettent au défi les meilleurs systèmes de faire encore plus avec moins. Les exigences extrêmes d'aujourd'hui peuvent être grossièrement regroupées en deux catégories qui ont un impact sur deux niveaux différents d'utilisations :

- Fournir des plateformes agiles permettant aux développeurs d'applications d'exceller
- Prise en charge d'architectures de systèmes évolutives plus performantes que les autres

Développer avec souplesse

Pour rester compétitives dans l'économie numérique, les entreprises doivent innover — et maintenant elles doivent le faire plus rapidement que jamais. Cette innovation étant centrée sur le développement d'applications web, mobiles et IoT modernes, les développeurs sont soumis à une pression extraordinaire. La vitesse est essentielle, mais la souplesse l'est tout autant, car ces applications évoluent bien plus rapidement que les applications patrimoniales comme les ERP. Les bases de données relationnelles nécessitent une structure de données plates relativement restreinte qui ne répond pas bien aux changements fréquents du modèle data. Cela ne répond ni aux besoins des applications modernes, qui évoluent fréquemment, ni aux exigences des entreprises.

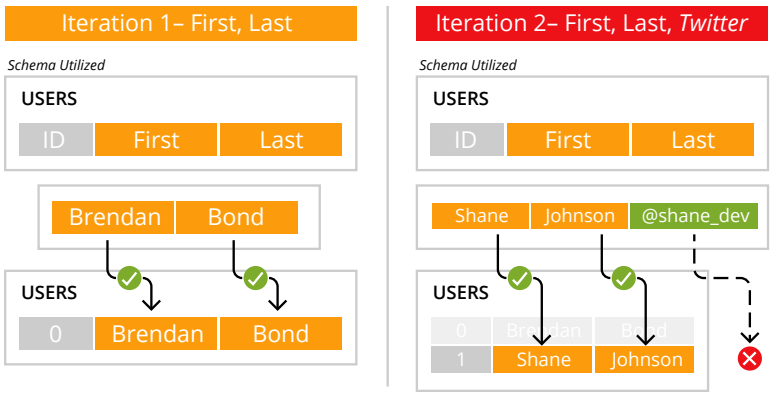
Toutes les plateformes agiles offrent une flexibilité qui permet un développement plus rapide et plus facile des applications. Certains de ces avantages résident dans la manière dont la plate-forme traite les données, d'autres dans la manière dont les applications peuvent interagir avec la base de données.

Exigences en matière de schémas NoSQL adaptables

L'un des principes fondamentaux du développement agile est l'adaptation à l'évolution des exigences de l'application : lorsque les exigences changent, le modèle de données change également. C'est un problème pour les bases de données relationnelles car le modèle de données est fixe et défini par un schéma statique. Ainsi, pour changer le modèle de données, les développeurs doivent modifier le schéma ou, pire, demander un "changement de schéma" aux administrateurs de la base de données. Cela ralentit ou arrête le développement, car il s'agit d'un processus manuel et chronophage, impactant les autres applications et services.

FIGURE 1

SGBDR – Un schéma explicite empêche l'addition de nouvelles attributions sur demande.

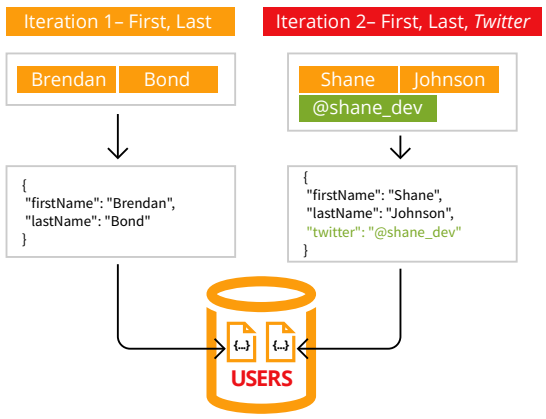


Flexibilité pour un développement plus rapide

En comparaison, une base de données documentaire NoSQL soutient pleinement le développement agile, car elle est sans schéma et ne définit pas statiquement comment les données doivent être modélisées. Au lieu de cela, elle s'en remet aux applications et aux services, et donc aux développeurs, pour savoir comment les données doivent être modélisées. Avec NoSQL, le modèle de données est défini par le modèle d'application. Les applications et les services modélisent les données comme des objets.

FIGURE 2

JSON – Le modèle data évolue à mesure que de nouvelles attributions sont ajoutées sur demande.



Simplicité pour faciliter le développement

Les applications et les services modélisent les données sous forme d'objets (par exemple, le profil d'un employé), les données à valeurs multiples sous forme de collections (par exemple, les rôles) et les données liées sous forme d'objets ou de collections imbriqués (par exemple, la relation avec le manager). Cependant, les bases de données relationnelles modélisent les données sous forme de tableaux de lignes et de colonnes — les données liées sous forme de lignes dans des tableaux différents, les données à valeurs multiples sous forme de lignes dans le même tableau. Le problème des bases de données relationnelles est que les données sont lues et écrites en décomposant, et en réassemblant les objets. C'est le "décalage d'impédance".

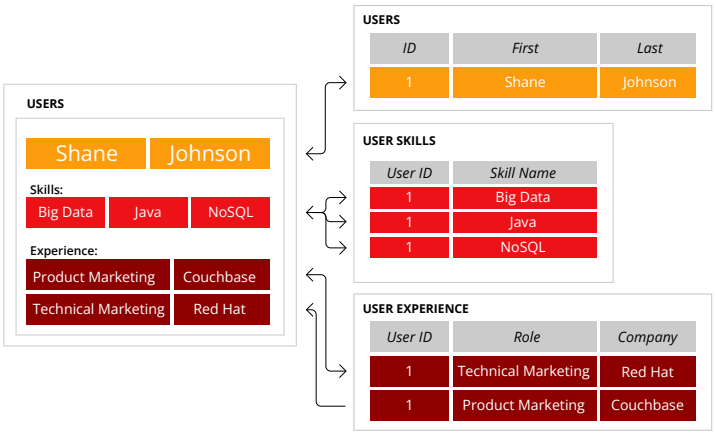
La solution de rechange consiste à utiliser des cadres de mappage objet-relationnel (ORM), qui sont, au mieux, inefficaces, au pire, problématiques.

Considérons une application de gestion de CV. Elle interagit avec les CV comme un objet d'objets utilisateurs. Elle contient un tableau pour les compétences et une collection pour les postes. Cependant, l'écriture d'un CV dans une base de données relationnelle nécessite que l'application décompose l'objet utilisateur.

Pour stocker ce CV, l'application doit insérer six lignes dans trois tableaux, comme l'illustre la **figure 3**.

FIGURE 3

SGBDR – Objets d'applications « shred » collectés dans des rangées de données, stockées dans des tableaux multiples.



Toutefois, la lecture de ce profil obligerait l'application à lire six lignes de trois tableaux, comme l'illustre la **figure 4**.

FIGURE 4

SGBDR – Les interrogations renvoient des données doublons, les applications doivent donc les filtrer.

Shane	Johnson	Big Data	Product Mktg	Couchbase
Shane	Johnson	Big Data	Technical Mktg	Red Hat
Shane	Johnson	Java	Product Mktg	Couchbase
Shane	Johnson	Java	Technical Mktg	Red Hat
Shane	Johnson	NoSQL	Product Mktg	Couchbase
Shane	Johnson	NoSQL	Technical Mktg	Red Hat

En revanche, une base de données NoSQL orientée documents lit et écrit des données formatées en JSON — qui est la norme de facto pour la consommation et la production de données pour le web, le mobile, et les applications IoT. Elle ne se contente pas d'éliminer le décalage d'impédance objet- relationnel : elle élimine la surcharge des cadres ORM et simplifie le développement des applications. Et pour cause, les objets sont lus et écrits sans être "déchiquetés", c'est-à-dire qu'un seul objet peut être lu ou écrit comme un seul document, comme l'illustre la **figure 5**.

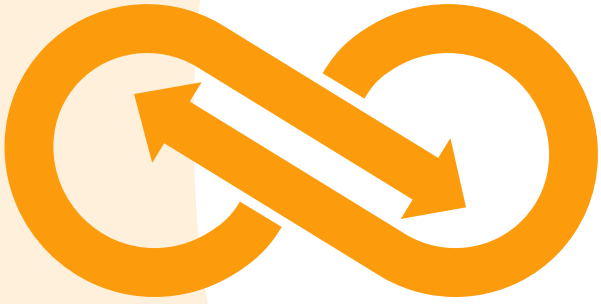
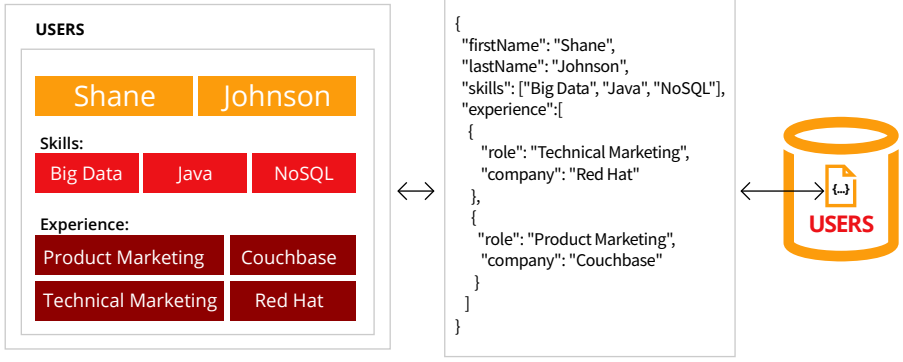


FIGURE 5

JSON – Les applications peuvent stocker des objets avec des données imbriquées dans un seul et unique document.



Regroupement des documents pour en faciliter l'accès

Contrairement à l'utilisation d'ensembles prédéfinis de schémas pour différencier les tables les unes des autres, les bases de données NoSQL disposent d'un concept, tel que les buckets, qui servent de zone de stockage générale pour tous les documents. Une base de données peut avoir de nombreux compartiments logiques, nommés, à des fins diverses. Le nom est fourni lors de la connexion ou de la demande de données et permet aux applications de disposer de leur propre zone dans le système.

À l'intérieur de ces bacs se trouvent des regroupements logiques hiérarchiques supplémentaires qui peuvent être limités à des utilisateurs ou des rôles particuliers. Ces regroupements sont appelés collections et/ou scopes, permettant de nommer des sous-ensembles de documents dans un bucket. Comme cette flexibilité permet de séparer les données d'un utilisateur ou d'une application d'une autre, le développeur n'a pas besoin de créer son propre code de sécurité et de fiabilité, mais peut laisser la base de données sous-jacente s'en charger.

Interrogation à l'aide de SQL

Les développeurs d'applications qui ont l'habitude d'effectuer des requêtes en SQL peuvent continuer à utiliser le même langage dans les plateformes NoSQL, mais en opérant sur les données JSON qui sont stockées. Par exemple, Couchbase fournit un langage d'interrogation basé sur SQL, appelé N1QL, qui renvoie des résultats en JSON avec des ensembles de lignes et des composants de sous-documents, le cas échéant. Cela contraste avec la grande majorité des autres bases de données NoSQL (comme MongoDB™) qui n'utilisent pas le langage SQL et obligent les développeurs à gravir une nouvelle courbe d'apprentissage du langage.

Les instructions standard sont prises en charge, notamment la syntaxe SELECT .. FROM .. WHERE. N1QL supporte également l'agrégation, le tri et les jointures (GROUP BY .. SORT BY .. LEFT OUTER/INNER JOIN). L'interrogation de collections, de scopes et même de tableaux imbriqués est prise en charge. Les performances des requêtes peuvent être améliorées avec des index composites, partiels, couvrants, etc.

Les changements sont minimes pour les experts SQL qui souhaitent passer à N1QL, toutes les requêtes de base fonctionnant d'emblée.

SQL	N1QL
SELECT p.FirstName + ' ' + p.LastName AS Name, d.City FROM AdventureWorks2016.Person.Person AS p INNER JOIN AdventureWorks2016.HumanResources.Employee e ON p.BusinessEntityID = e.BusinessEntityID INNER JOIN (SELECT bea.BusinessEntityID, a.City FROM AdventureWorks2016.Person.Address AS a INNER JOIN AdventureWorks2016.Person.BusinessEntityAddress AS bea ON a.AddressID = bea.AddressID) AS d ON p.BusinessEntityID = d.BusinessEntityID ORDER BY p.LastName, p.FirstName;	SELECT p.FirstName || ' ' || p.LastName AS Name, d.City FROM AdventureWorks2016.Person.Person AS p INNER JOIN AdventureWorks2016.HumanResources.Employee e ON p.BusinessEntityID = e.BusinessEntityID INNER JOIN (SELECT bea.BusinessEntityID, a.City FROM AdventureWorks2016.Person.Address AS a INNER JOIN AdventureWorks2016.Person.BusinessEntityAddress AS bea ON a.AddressID = bea.AddressID) AS d ON p.BusinessEntityID = d.BusinessEntityID ORDER BY p.LastName, p.FirstName;

Qu'en est-il des transactions ACID en NoSQL ?

Les bases de données NoSQL fonctionnent également comme des systèmes opérationnels avec un grand nombre de transactions. Lorsque vous aplatissez une entité commerciale en plusieurs tables distinctes, vous devez effectuer une transaction pour presque chaque mise à jour. Avec les bases de données NoSQL, vous n'avez pas besoin d'aplatir l'entité, mais pouvez généralement la contenir dans un document unique. Les mises à jour d'un document unique sont atomiques et ne nécessitent pas de transaction. Cependant, il peut y avoir des mises à jour qui couvrent plusieurs documents et qui nécessitent une vérification pour s'assurer que "tout ou rien" de la transaction se produit.

C'est pourquoi les bases de données NoSQL comme Couchbase prennent en charge les transactions.

Exemple Java :

```
Transactions transactions = Transactions.create(cluster,
      TransactionConfigBuilder.create()
.durabilityLevel(TransactionDurabilityLevel.PERSIST_TO_MAJORITY)
            .logOnFailure(true, Event.Severity.WARN)
      .build());
TransactionResult result = transactions.run((ctx) -> {
      // Inserting a doc:
      ctx.insert(collection, "doc-a", JsonObject.create());
      // Getting documents:
      // Use ctx.getOptional if the document may or may not exist
      Optional<TransactionGetResult> docOpt =
            ctx.getOptional(collection, "doc-a");
      // Use ctx.get if the document should exist, and the transaction
      // will fail if it does not
      TransactionGetResult docA = ctx.get(collection, "doc-a");
      // Replacing a doc:
      TransactionGetResult docB = ctx.get(collection, "doc-b");
      JsonObject content = docB.contentAs(JsonObject.class);
      content.put("transactions", "are awesome");
      ctx.replace(docB, content);
      // Removing a doc:
      TransactionGetResult docC = ctx.get(collection, "doc-c");
      ctx.remove(docC);
      ctx.commit();
});
```

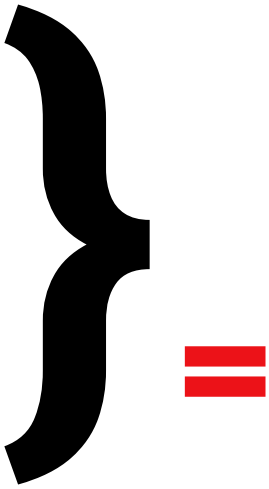
La combinaison des transactions et du SQL élargit considérablement le nombre de cas d'utilisation pour lesquels une base de données NoSQL peut être envisagée. Dans le passé, l'incapacité de joindre ou de traiter des opérations transactionnelles signifiait que les bases de données NoSQL n'étaient choisies que pour les cas d'utilisation à volume et échelle élevés. Mais la possibilité d'utiliser SQL et les transactions signifie que les bases de données NoSQL peuvent également être choisies pour les cas traditionnels de SGBDR qui nécessitent plus de flexibilité et de puissance.

De plus, en choisissant une base de données NoSQL transactionnelle, de nombreuses applications traditionnellement complexes peuvent être simplifiées car elles n'ont pas besoin d'un outil ORM.

Une seule source de données — plusieurs méthodes d'accès

Les bases de données NoSQL fonctionnent comme un magasin de contenu primaire, ce qui signifie que vous saisissez les données dans une seule application mais que vous pouvez y accéder de plusieurs manières en fonction du cas d'utilisation. Par exemple, les développeurs peuvent utiliser des appels API directs pour accéder à un document spécifique à l'aide d'une clé ou par le biais d'une requête SQL qui renvoie plusieurs lignes de données dans une réponse JSON.

D'autres méthodes d'accès sont disponibles en fonction de la base de données, notamment des systèmes de recherche en texte intégral qui permettent des demandes de recherche en langage naturel. Les demandes peuvent porter sur des correspondances "floues" complètes ou partielles, des plages géographiques ou des recherches avec caractères génériques. La réponse comprend un document JSON avec des listes d'identifiants de documents correspondants, des informations contextuelles et un score de pertinence.



Les systèmes de recherche en texte intégral sont souvent séparés d'une base de données, mais les bases de données NoSQL comme Couchbase les intègrent au système sousjacent, ce qui permet aux gestionnaires de simplifier l'architecture globale.

Les analyses de données volumineuses sont également possibles, en utilisant des sous-systèmes complémentaires qui traitent des volumes plus importants de données historiques. En utilisant des capacités d'indexation et d'interrogation avancées, basées sur un langage connu sous le nom de SQL++, des analyses plus poussées peuvent être effectuées dans la même base de données sans avoir besoin d'un système OLAP externe.

Les données étant stockées et indexées au sein d'une même base de données, les développeurs peuvent se connecter à un seul système et transmettre les requêtes pertinentes.

Fonctionner à n'importe quelle échelle

Les bases de données qui prennent en charge les applications web, mobiles et IoT doivent être capables de fonctionner à n'importe quelle échelle. S'il est possible de faire évoluer une base de données relationnelle comme Oracle (en utilisant, par exemple, Oracle RAC), cette opération est généralement complexe, coûteuse et n'est pas totalement fiable. Avec Oracle, par exemple, la mise à l'échelle à l'aide de la technologie RAC nécessite de nombreux composants et crée un point de défaillance unique qui met en péril la disponibilité.

En comparaison, une base de données distribuée NoSQL — conçue avec une architecture scale-out et sans point de défaillance unique — offre des avantages opérationnels convaincants.

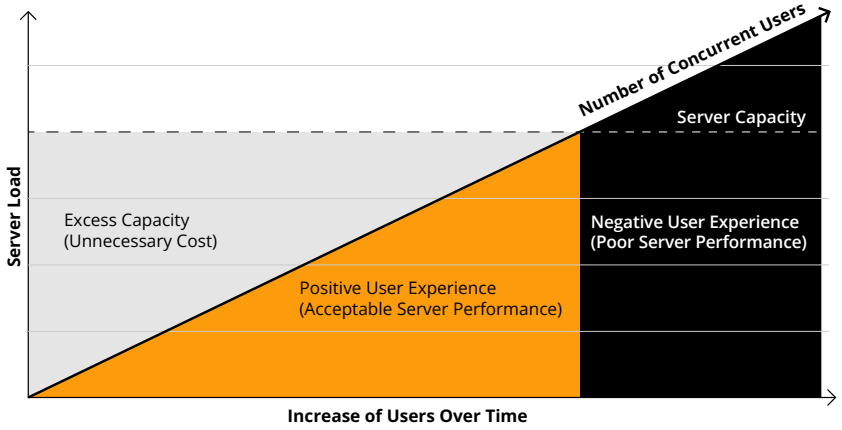
L'élasticité pour des performances à grande échelle

Les applications et les services doivent prendre en charge un nombre toujours croissant d'utilisateurs et de données — des centaines, des milliers, voire des millions d'utilisateurs, et des giga-octets, voire des téraoctets de données opérationnelles. Dans le même temps, ils doivent évoluer pour maintenir les performances, et ce de manière efficace.

La base de données doit être capable de faire évoluer les lectures, les écritures et le stockage. C'est un problème pour les bases de données relationnelles qui sont limitées à la mise à l'échelle, c'est-à-dire qu'elles ne peuvent évoluer qu'en ajoutant plus de processeurs, de mémoire et de stockage à un seul serveur physique. Par conséquent, la capacité à évoluer efficacement et à la demande est un défi. Elle devient de plus en plus coûteuse, car les entreprises doivent acheter des serveurs de plus en plus grands pour accueillir davantage d'utilisateurs et de données. En outre, cela peut entraîner des temps d'arrêt si la base de données doit être mise hors ligne pour effectuer des mises à niveau matérielles.

FIGURE 6

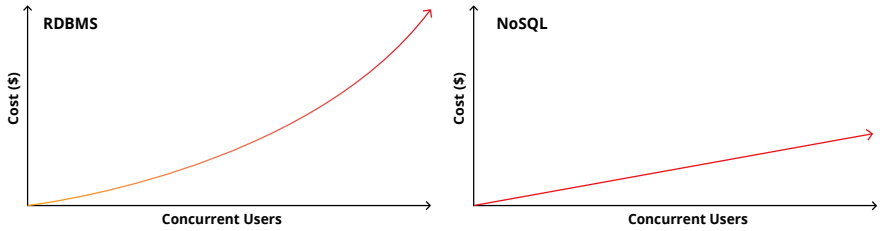
SGBDR – Le serveur est trop grand ou trop petit, amenant des coûts inutiles ou de faibles performances.



En revanche, une base de données NoSQL distribuée exploite le matériel de base pour évoluer, c'est-à-dire pour ajouter des ressources supplémentaires en ajoutant simplement des serveurs à un cluster. La capacité d'extension permet aux entreprises de s'adapter plus efficacement : a) en ne déployant pas plus de matériel que nécessaire pour répondre à la charge actuelle ; b) en exploitant du matériel moins coûteux et/ou une infrastructure Cloud ; et c) en s'adaptant à la demande et sans temps d'arrêt.

FIGURE 7

NoSQL – Ajouter des serveurs de grande diffusion sur demande, afin que les ressources matérielles correspondent à la charge d'application.



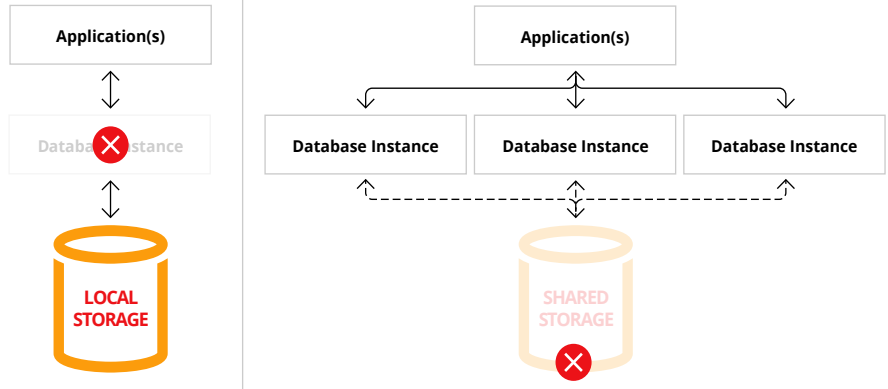
En distribuant les lectures, les écritures et le stockage sur un cluster de noeuds, les bases de données NoSQL sont capables de fonctionner à n'importe quelle échelle. De plus, elles sont conçues pour être faciles à configurer, à installer et à gérer, qu'il s'agisse de petits ou de grands clusters.

Disponibilité pour un déploiement global et permanent

Comme de plus en plus d'engagements des clients se font en ligne via des applications web et mobiles, la disponibilité devient une préoccupation majeure, voire primordiale. Ces applications critiques doivent être disponibles 24 heures sur 24, 7 jours sur 7, sans exception. Assurer une disponibilité 24 heures sur 24 et 7 jours sur 7 est un défi pour les bases de données relationnelles qui sont déployées sur un seul serveur physique ou qui reposent sur un clustering avec un stockage partagé. Si elles sont déployées sur un seul serveur et que celui-ci tombe en panne, ou sur un cluster et que le stockage partagé tombe en panne, la base de données devient indisponible, les applications s'arrêtent et les clients se désengagent.

FIGURE 8

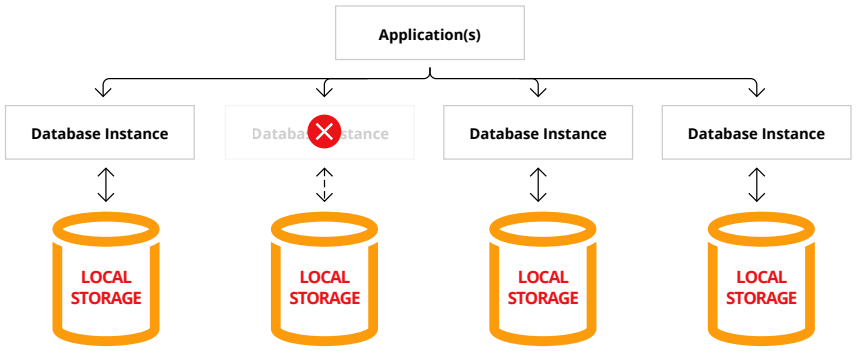
SGBDR – L'échec d'un serveur ou d'un appareil de stockage met en panne la base de données toute entière.



Contrairement à la technologie relationnelle, une base de données distribuée, NoSQL, partitionne et distribue les données à plusieurs instances de base de données sans ressources partagées. En outre, les données peuvent être répliquées sur une ou plusieurs instances pour une haute disponibilité (réplication inter-clusters) et dans différents lieux géographiques. Alors que les bases de données relationnelles comme Oracle nécessitent un logiciel distinct pour la réplication, par exemple Oracle Active Data Guard, les bases de données NoSQL n'en ont pas besoin — elle est intégrée et automatique. En outre, le basculement automatique garantit que si un noeud tombe en panne, la base de données peut continuer à effectuer des lectures et des écritures en envoyant les demandes à un autre noeud.

FIGURE 9

NoSQL – Si une instance échoue, l'application peut envoyer des requêtes à une autre application.

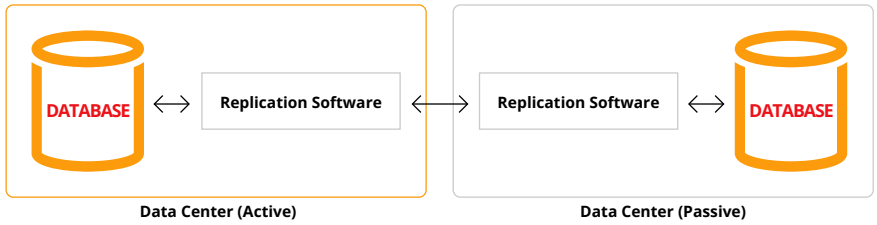


Le comportement des clients exige désormais des organisations qu'elles prennent en charge plusieurs canaux physiques, en ligne et mobiles dans plusieurs régions et souvent plusieurs pays. Si le déploiement d'une base de données dans plusieurs data-centers augmente la disponibilité et facilite la reprise après sinistre, il présente également l'avantage d'améliorer les performances. Toutes les lectures et écritures peuvent être exécutées sur le centre de données le plus proche, ce qui réduit la latence.

Il est difficile d'assurer une disponibilité globale pour les bases de données relationnelles lorsque des modules complémentaires distincts sont nécessaires — ce qui accroît la complexité — ou lorsque la réplication entre plusieurs data-centers ne peut être utilisée que pour le basculement, car un seul data-center est actif à la fois. Oracle, par exemple, nécessite Oracle GoldenGate. Lors de la réplication entre data-centers, les applications basées sur des bases de données relationnelles peuvent subir une dégradation des performances ou constater que les data-centers sont fortement désynchronisés.

FIGURE 10

SGBDR – Nécessite un logiciel annexe pour copier des données dans d'autres data-centers.

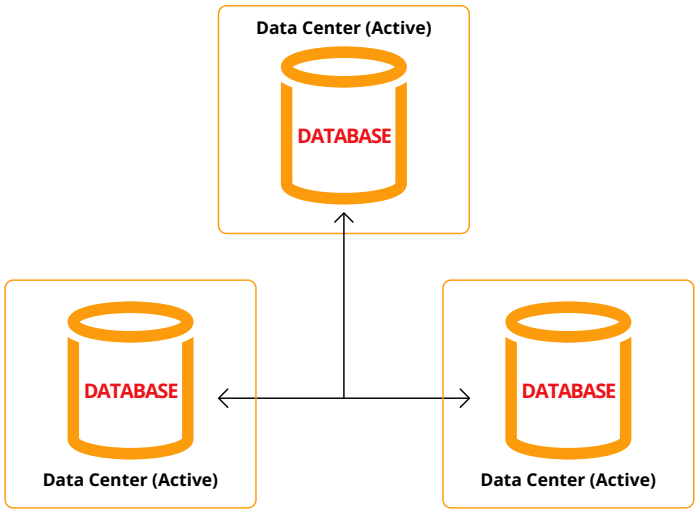


Une base de données NoSQL distribuée comprend une réplication intégrée entre les data-centers — aucun logiciel distinct n'est nécessaire. En outre, certaines bases NoSQL incluent une réplication bidirectionnelle permettant des déploiements actifs-actif complets vers plusieurs data-centers. Cela permet de déployer la base de données dans plusieurs pays ou régions tout en fournissant un accès aux données locales aux applications locales et à leurs utilisateurs.

Le déploiement dans plusieurs data-centers améliore non seulement les performances, mais permet également un basculement immédiat via des routeurs matériels. Les applications n'ont pas besoin d'attendre que la base de données détecte la panne et effectue son propre basculement.

FIGURE 11

NoSQL – Replication between data centers is fully built-in and can be bidirectional.



Le NoSQL est mieux adapté aux besoins de l'économie numérique

Alors que les entreprises se tournent vers l'économie numérique — rendue possible par les technologies du Cloud, du mobile, des réseaux sociaux et du big data — les développeurs et les équipes d'exploitation doivent créer et maintenir des applications web, mobiles et IoT de plus en plus rapidement, et à plus grande échelle. Le NoSQL est de plus en plus la technologie de base de données privilégiée pour alimenter les applications web, mobiles et IoT d'aujourd'hui.

Des centaines d'entreprises du Global 2000, ainsi que des dizaines de milliers de petites entreprises et de startups, ont adopté le NoSQL. Pour beaucoup d'entre elles, l'utilisation de NoSQL a commencé par un cache open source, une preuve de concept ou une petite application, puis s'est étendue à des applications critiques ciblées, et constitue désormais le fondement de tout développement d'application. Aujourd'hui, la base de données NoSQL Couchbase sert des milliers de clients de ce type.

Avec le NoSQL, les entreprises sont mieux à même de développer avec agilité et de fonctionner à n'importe quelle échelle — et de fournir les performances et la disponibilité requises pour répondre aux exigences des entreprises de l'économie numérique.

*

Modern customer experiences need a flexible database platform that can power applications spanning from cloud to edge and everything in between. Couchbase's mission is to simplify how developers and architects develop, deploy and consume modern applications wherever they are. We have reimagined the database with our fast, flexible and affordable cloud database platform Capella, allowing organizations to quickly build applications that deliver premium experiences to their customers—all with best-in-class price performance. More than 30% of the Fortune 100 trust Couchbase to power their modern applications.

For more information, visit www.couchbase.com and follow us on Twitter.