Microsoft

Montée en charge par ajout de serveurs de SQL Server

Tour d'horizon des techniques permettant la montée en charge par ajout de serveurs des applications de bases de données SQL Server.

Livre blanc SQL Server

Date de publication : mars 2012

Sujet : SQL Server 2005, SQL Server 2008, SQL Server 2008 R2, SQL Server 2012

Introduction

Ce livre blanc décrit les différentes technologies qui permettent la montée en charge par ajout de serveurs (« scale out ») d'une application de base de données Microsoft® SQL Server®, en détaillant les facteurs de choix de la ou des solutions adaptées à votre application. **On parle également de montée en charge horizontale**.

Copyright

Table des matières

La montée en charge horizontale ou « scale out »

L'évolutivité est la capacité d'une application à exploiter efficacement davantage de ressources pour faire face à une charge de travail plus importante. Par exemple, une application utilisée par quatre personnes sur un système doté d'un seul processeur peut desservir 15 utilisateurs sur un système à quatre processeurs. On parle alors d'une application évolutive. Si l'ajout de processeurs n'augmente pas le nombre d'utilisateurs desservis (par exemple, avec une application à thread unique), l'application n'est pas évolutive.

Il existe deux types d'évolutivité : la montée en charge verticale (« scale up ») et la montée en charge horizontale (« scale out »). La montée en charge verticale consiste à évoluer vers un serveur plus volumineux et plus puissant, par exemple en passant d'un système à quatre processeurs à un système doté de 128 processeurs. Pour une base de données, c'est la méthode d'évolution la plus couramment utilisée. Lorsque votre base de données n'a plus suffisamment de ressources sur le matériel utilisé, il suffit de vous procurer un système plus important, équipé de davantage de processeurs et de mémoire. Point positif de la montée en charge verticale, elle ne nécessite pas de modifications significatives dans la base de données. En général, il suffit d'installer la base de données sur un système plus puissant et de continuer à l'utiliser normalement, l'augmentation de puissance permettant de gérer une charge de travail plus importante. La montée en charge horizontale consiste à développer plusieurs serveurs au lieu de disposer d'un seul serveur plus conséquent. Cette démarche est intéressante du point de vue du coût initial du matériel : huit serveurs équipés de quatre processeurs reviennent généralement moins cher qu'un seul serveur de 32 processeurs. Mais cet avantage est souvent balayé par les frais de licence et de maintenance. Dans certains cas, la redondance d'une montée en charge horizontale peut également s'avérer utile en termes de disponibilité.

D'innombrables articles, ouvrages et livres blancs ont déjà été publiés sur les technologies de montée en charge horizontale de SQL Server. Le présent livre blanc s’intéresse plus particulièrement aux facteurs à prendre en considération pour choisir la solution la plus pertinente pour votre application. Pour plus d'informations sur les technologies de montée en charge horizontale, reportez-vous à la section « Références ».

Types de données

Pour choisir la meilleure stratégie de montée en charge horizontale, il est essentiel de comprendre que les applications gèrent différents types de données, chacun impliquant des exigences spécifiques en termes d'architecture de montée en charge. Avec une seule base de données, le plus simple est d'appliquer le même traitement à toutes les données. Mais lorsque l'on commence à répartir et à répliquer les données en vue d'une montée en charge horizontale, il est important de bien comprendre leur utilisation pour adopter la solution la plus adaptée. Dans nombre d'applications, plusieurs approches de montée en charge horizontale sont requises en raison des multiples types de données impliqués. Cette section aborde trois types de données et leur incidence sur les décisions de montée en charge horizontale.

Données de référence

Les données de référence, comme leur nom l'indique, sont des informations utilisées par une application qui ne gère pas leur maintenance. Les données de référence sont relativement stables et leur validité s'étend sur une période précise. Parmi les exemples représentatifs de données de référence, citons les catalogues de pièces utilisés par les systèmes de saisie de commandes, les horaires des compagnies aériennes ou encore les plans comptables exploités par les systèmes financiers. La stabilité relative de ces données s'explique par le fait qu'elles peuvent être utilisées par d'autres applications. Par conséquent, des changements trop fréquents pourraient générer une certaine confusion. Ainsi, un prix catalogue modifié plusieurs fois dans une même journée risque de provoquer l'incompréhension et le mécontentement des clients (bien sûr, cela ne s'applique pas au cours des actions ni aux prix des carburants qui évoluent en permanence). Les données de référence changent habituellement à intervalles fixes. Par exemple, les prix peuvent évoluer chaque jour ou chaque semaine, et les numéros de compte chaque mois. Les données de référence peuvent également comporter un indicateur de version qui apparaît dans les transactions faisant appel à ces données. Par exemple, un bon de commande peut faire référence à la version du catalogue utilisée pour la création de la commande, ce qui évite toute ambiguïté sur le mode de détermination des prix. Une entreprise peut faire le choix d'accepter plusieurs versions de données de référence pour être certaine que ses clients ne se basent pas sur des données obsolètes.

Les données de référence restant stables sur une certaine période et souvent identifiées par un numéro d'ordre qui indique la version utilisée, il est possible de les copier sur de nombreux systèmes différents, sans risque majeur d'incohérence. Dans une batterie de serveurs Web, chaque entité doit héberger un exemplaire du catalogue pour pouvoir répondre rapidement aux requêtes de consultation du catalogue. Le plus souvent, les données de référence communes sont mises en cache dans la mémoire. Les données de référence stables peuvent être transférées vers un client intelligent pour offrir un accès rapide et la possibilité de naviguer hors ligne. En cas de perte ou d'altération de la copie, il est facile d'en obtenir une nouvelle. Le numéro de version des données de référence permet de déterminer s'il existe une version plus récente disponible.

Toutes les copies des données de référence sont celles d'une copie principale. Elles ne sont donc actualisées qu'en cas de modification du fichier maître. Ce système évite les conflits de mise à jour. Ainsi, toute réplication de capture instantanée ou transactionnelle peut être utilisée pour tenir à jour les données de référence. Le détenteur de la copie principale peut demander à un service de renvoyer une copie des données de référence aux applications qui ne les stockent pas dans une base de données.

En conclusion, la montée en charge horizontale des données de référence est facile à mettre en œuvre et peut améliorer les performances avec un investissement minimum. D'autres types de données courants présentent les mêmes caractéristiques et peuvent être traités comme les données de référence. Ainsi, l'histoire ne change pour ainsi dire jamais (même si elle se répète souvent). C'est pourquoi, les données historiques type chiffres trimestriels des ventes ou cours des actions peuvent être traitées comme des données de référence.

Données d'activité

Les données d'activité sont des informations associées à une activité particulière ou à une transaction commerciale. Par exemple, un bon de commande ou la vente d'un produit en stock génère des données liées à la transaction. Ces données entrent dans le champ d'application d'une activité commerciale et, sauf pour les besoins de la conservation d'un historique, elles n'ont plus grande utilité une fois l'activité terminée. Au terme de l'activité, les données correspondantes deviennent généralement des données de référence, et les mêmes considérations s'appliquent en matière de montée en charge.

En général, les données d'activité présentent de faibles exigences de simultanéité. En effet, il est peu probable que plusieurs centaines d'utilisateurs tentent d'accéder au même moment à un même bon de commande. Bien que ce principe ne soit pas universel, il est suffisamment fiable pour servir de base à la stratégie de montée en charge des données d'activité. Les données d'activité possèdent par ailleurs des taux de mise à jour raisonnablement faibles. Par exemple, lorsqu'un bon de commande est créé, il n'est modifié que par des événements de type changement de statut ou date de livraison, ce qui arrive relativement rarement (plusieurs fois par jour, par rapport à de nombreuses actualisations chaque seconde). Les données d'activité sont généralement faciles à identifier sans ambiguïté et ne sont pas souvent consultées en dehors du cadre de l'activité. Ce qui veut dire que si la montée en charge implique la répartition des données d'activité entre plusieurs bases de données, elles resteront faciles à trouver. Une transaction commerciale devant accéder à un bon de commande connaît le plus souvent son numéro. Par conséquent, partitionner les bons de commandes par plage de numéro permet d'identifier facilement la base de données qui contient le bon de commande recherché.

Les données d'activité entrent souvent dans le champ d'application d'un autre objet de données. Il peut donc être judicieux de stocker toutes les commandes passées par un client dans la même base de données avec les autres informations relatives à ce client, si ce mode d'accès aux commandes est le plus courant. De même, vous pouvez stocker tous les bons de commandes d'un fournisseur dans la base de données de ce fournisseur.

En conclusion, il est relativement facile de répliquer des données d'activité lorsque nécessaire et, dans de nombreux cas, il est pratique de partitionner les données entre plusieurs bases de données pour les besoins d'une montée en charge horizontale. La méthode de montée en charge horizontale à retenir pour les données d'activité dépend de l'utilisation des données, comme l'explique la section « Facteurs influant sur la montée en charge horizontale ».

Données de ressources

Les données relatives aux ressources sont les informations centrales dont dépend votre entreprise : inventaire, informations comptables, fichier de clientèle, etc. Une perte de données de ressources peut purement et simplement stopper votre activité. Les bases de données utilisent de nombreuses fonctionnalités pour garantir l'intégrité et la haute disponibilité des données afin que ces données sensibles soient constamment disponibles. Elles imposent généralement de grandes exigences en matière de simultanéité car de nombreuses applications et utilisateurs différents doivent y accéder. Elles possèdent

également un taux de mise à jour élevé. La quantité disponible pour un article ou le solde d'un compte sont des valeurs qui changent plusieurs fois par jour. Si le grand nombre de mises à jour simultanées fait de la montée en charge horizontale une option très intéressante en termes de performances, le besoin d'intégrité et de haute disponibilité de ces données implique bien souvent une montée en charge verticale.

Les données relatives aux ressources ne représentent en général que les données actives. Les comptes inactifs et les pièces dont la fabrication n'est pas continue, par exemple, sont souvent conservés dans des tableaux historiques relativement statiques, où ils deviennent de fait des données de référence. Des captures instantanées des données de ressources peuvent être enregistrées pour la création de rapports ou d'un historique ; il s'agit là aussi de données de référence. Les exigences d'intégrité et de haute disponibilité inhérentes à ce type de données perdent de leur importance lorsque ces informations deviennent des données de référence. Cette conversion en données de référence préserve la pertinence des données de ressources, limite leur volume et réduit la nécessité d'une montée en charge.

Facteurs influant sur la montée en charge horizontale

Nous venons de voir les types de données stockées dans des systèmes de base de données. Intéressons-nous à présent aux caractéristiques d'utilisation qui vont influer sur le choix de la technologie de montée en charge pour ces données. SQL Server prend en charge plusieurs technologies de montée en charge horizontale. Pour choisir la plus adaptée, il faut avant tout étudier les particularités des données et de l'application concernées.

Fréquence de mise à jour

Certaines données sont actualisées très fréquemment. C'est le cas des données de journaux issues des serveurs Web et des mesures d'instruments émises par les machines d'un atelier (remarque : ces données sont plus concernées par des processus d'insertion que de mise à jour, mais pour les besoins de notre démonstration, nous considèrerons que c'est la même chose). D'autres données sont rarement, voire jamais, mises à jour et peuvent être considérées comme des informations essentiellement en lecture seule. C'est le cas des données historiques telles que les chiffres trimestriels des ventes. Les informations conservées dans des entrepôts de données sont généralement mises à jour par lots en dehors des horaires de travail, mais leur statut passe en lecture seule lorsque l'entrepôt est utilisé.

Il est assez difficile de répliquer efficacement les données mises à jour très fréquemment car la surcharge imposée par la réplication des actualisations limite l'évolutivité des copies répliquées. Autrement dit, la base de données passe tellement de temps à répliquer les changements que ses performances globales peuvent devenir inférieures à ce qu'elles seraient si les données étaient conservées dans une base de données unique. Il est donc essentiel de connaître le taux de mise à jour de vos données lorsque vous envisagez une réplication pour une montée en charge horizontale.

Une base de données avec un taux de mise à jour très faible est facile à répliquer. Dans ce cas, la réplication peut être une stratégie adaptée de montée en charge horizontale. Si le taux de mise à jour est suffisamment modéré pour que la base de données puisse être considérée en lecture seule la plupart du temps, le choix de la base de données partagée évolutive prend tout son intérêt. Cette option sera détaillée plus loin dans ce document.

Aptitude à modifier l'application

Bien que ce ne soit pas une caractéristique des données au sens strict, le degré de flexibilité dont vous disposez pour reconfigurer l'application peut avoir un impact considérable sur les stratégies de montée en charge horizontale adaptées à vos besoins. Certaines de ces stratégies ne requièrent aucun changement dans l'application ; certaines demandent des modifications mineures dans les requêtes et les procédures stockées. D'autres enfin peuvent nécessiter de repenser entièrement le fonctionnement de l'application. Naturellement, la flexibilité est maximale si vous partez d'une application entièrement nouvelle. C'est pourquoi, il est recommandé d'anticiper la montée en charge dès la phase de conception d'une application, même si cette opération est inutile au départ. En effet, apporter des modifications une fois que l'application est en production et manque de ressources s'avère beaucoup plus compliqué que l'intégration initiale d'une option de montée en charge horizontale. Les applications packagées ainsi que certaines applications existantes peuvent être impossibles à modifier. Dans ce cas, la montée en charge horizontale doit être absolument transparente pour le code de l'application.

Capacité de partitionnement des données

L'une des méthodes les plus efficaces pour monter les données en charge de manière horizontale consiste à les partitionner entre plusieurs bases de données de telle sorte que chaque serveur gère une partie des données. Bien qu'assez évidente, cette technique n'est pas efficace pour toutes les données. De plus, lorsqu'elle est applicable, le mode de partitionnement joue un rôle important sur les performances.

Pour illustrer les effets du partitionnement, voyons comment une base de données de commandes pourrait être partitionnée. Les commandes peuvent être partitionnées en fonction des produits commandés : livres dans une base de données, vêtements dans une autre, etc. Outre les questions courantes (Comment traiter une commande contenant à la fois des livres et des vêtements ?), ce modèle ne donne pas de bons résultats si la majorité des requêtes relie les commandes aux clients, car cette jonction nécessite de rechercher les commandes correspondantes dans toutes les bases de données de commandes.

Autre méthode de partitionnement d'une base de données de commandes : subdiviser les commandes par plages de numéros. Cette méthode peut être intéressante si les commandes sont majoritairement consultées par numéro plutôt que par emplacement. Mais si les jonctions avec la table des clients sont nombreuses, cette technique impose aussi de distribuer les jonctions. La seule solution à ce problème consisterait à partitionner la base de données des commandes par numéro de client, de telle sorte que la base de données de commandes à utiliser pour un client particulier soit toujours connue. Cette technique est particulièrement efficace si la base de données des clients est partitionnée, et si les

commandes de chaque client sont stockées dans la même base de données que le client. D'autres informations doivent être jointes aux données de commandes et, si possible, elles doivent être partitionnées selon le même modèle pour éviter la distribution de jonctions. Il peut s'agir de données de référence (descriptions d'article, par exemple) pouvant être répliquées vers toutes les bases de données de commandes pour éliminer la distribution de jonctions avec la base de données d'inventaire.

Si les données d'application peuvent être réparties entre plusieurs bases de données et si la puissance de traitement supplémentaire fournie par les différents serveurs dépasse les coûts de communication imputables au regroupement des résultats, les données peuvent être partitionnées. Il est impossible de partitionner efficacement toutes les données d'application, et il est crucial de bien choisir le modèle de partitionnement pour garantir l'efficacité de la montée en charge horizontale des données partitionnées.

Interdépendance et couplage des données

Une autre méthode pour distribuer les données en vue d'une montée en charge horizontale consiste à les subdiviser en fonction de leur utilisation. Si des parties de la base de données sont exploitées par différentes applications, il peut être intéressant de la subdiviser à la frontière des applications, de façon à ce que chacune dispose d'un traitement adapté aux données qu'elle utilise. Cette procédure est efficace lorsque les données exploitées par les différentes applications peuvent être segmentées pour permettre un traitement spécifique à la base de données sans que l'interdépendance des données ne pénalise le trafic réseau. À titre d'exemple extrême, si une base de données de commandes est subdivisée de telle sorte que les lignes de commande se trouvent dans une base de données tandis que tous les en-têtes de commande sont stockés dans une autre, chaque requête d'accès aux commandes devra effectuer une jonction distribuée entre les deux bases de données. Le trafic réseau généré par les jonctions distribuées annulera tous les avantages de la répartition des données.

Le meilleur moyen de déterminer comment subdiviser une base de données consiste à étudier soigneusement le modèle de données. Dans un schéma entité-relation, les relations représentent à la fois les chemins de jonction et les contraintes d'intégrité référentielles. Les jonctions et les contraintes sont coûteuses à mettre en œuvre dans des bases de données distribuées. La base de données ne mettra pas en application de contraintes référentielles entre les bases de données. Par conséquent, si des tables associées sont réparties entre plusieurs bases de données, la contrainte devra être ignorée ou appliquée par un déclencheur ou par l'application. Pour éviter ces problèmes, recherchez les îlots de données associées dans votre modèle de données. Ce sont des groupes de tables présentant peu ou pas de liens avec le reste du modèle de données. Dans une base de données de saisies de commandes, on trouve généralement peu de relations entre les tables de clients et les tables d'inventaire, par exemple.

Une fois que vous avez sélectionné les groupes de tables qui peuvent être isolés dans des bases de données différentes sans trop de problèmes d'intégrité référentielle, il convient d'étudier les modèles de mise à jour. Si la répartition de vos données entre les bases de données crée un grand nombre de transactions distribuées couvrant les bases de données,

la surcharge de mise à jour imposée par la validation en deux phases peut annuler certains avantages de la montée en charge horizontale.

Le dernier facteur à considérer pour le couplage de données est le mode de prise en charge des tables partagées. Un certain nombre de tables est accédé par plusieurs applications. Il est donc important, lorsque vous répartissez les données, de décider de l'emplacement des tables partagées. Parfois, une table peut être lue par plusieurs applications, mais mise à jour par une seule. Il est donc plus logique de la situer au même emplacement que l'application de mise à jour. Les problèmes d'intégrité relationnelle et de distribution évoqués plus haut peuvent également influencer le choix de l'emplacement de la table partagée. Si la table est relativement petite et utilisée intensivement par plusieurs applications, il est utile de la répliquer dans plusieurs bases de données. C'est la méthode la plus simple si la table est mise à jour par une seule application, car la réplication transactionnelle depuis la copie principale peut alors être utilisée pour maintenir à jour les autres copies.

Les données de ressources sont souvent extrêmement interdépendantes, et présentent de nombreuses contraintes d'intégrité, ce qui induit un niveau élevé de couplage. Dans certains cas, cela empêche de répartir les données de ressources par application, à moins de modifier l'application. Vous pouvez monter en charge les données de référence et d'activité de manière horizontale, ou partitionner les données de ressources de telle sorte que la montée en charge reste possible si les données de ressources sont étroitement couplées. Mais certaines options de montée en charge horizontale nécessitant la répartition des données peuvent imposer une redéfinition de l'architecture de l'application pour l'adapter à la nouvelle architecture des données. Comme indiqué dans la section sur la modification des applications, concevoir le modèle de données de manière à ce qu'il puisse être subdivisé par la suite est une pratique à suivre pour élaborer une nouvelle application.

Solutions de montée en charge horizontale

À présent que nous avons défini les facteurs qui interviennent dans le choix d'une solution de montée en charge pour SQL Server, nous allons étudier chacune de ces solutions et les facteurs en faveur de chacune. Ce livre blanc n'entre pas dans le détail des solutions de montée en charge horizontale. Pour plus d'informations, reportez-vous aux articles mentionnés à la section « Références ».

Bases de données partagées évolutives

La solution de montée en charge horizontale la plus facile à mettre en œuvre dans SQL Server est celle des bases de données partagées évolutives. Vous créez une base de données sur un réseau SAN et huit instances de SQL Server fonctionnant sur différents serveurs liés à la base de données pour commencer à prendre en charge les requêtes. Il s'agit là de la solution classique de « disque partagé » où la puissance de traitement est montée en charge de manière horizontale, mais où une seule image disque des données est utilisée. À ce stade, les experts de SQL Server se demandent : « Et les verrouillages ? Je croyais que chaque instance de SQL Server conservait ses propres verrouillages dans sa mémoire. » C'est bien le cas. Chaque instance conserve ses propres verrouillages de base de données, et aucune ne connaît ceux des autres instances. Le seul moyen de faire

marcher ce système est de n'avoir aucun verrouillage. Les bases de données partagées évolutives ne fonctionnent alors que si elles sont reliées à une base de données en lecture seule. Ce qui signifie que les bases de données partagées évolutives sont très utiles pour les entrepôts de données ou les bases de données de création de rapports, mais qu'elles ne conviennent pas aux applications qui mettent à jour des données. Revenons aux caractéristiques des données : les bases de données partagées évolutives ne fonctionnent que si la fréquence de mise à jour est nulle. Ces données sont, par définition, historiques. Ce sont donc toutes des données de référence. La Figure 1 présente l'utilisation de bases de données partagées évolutives comme solution de montée en charge horizontale.

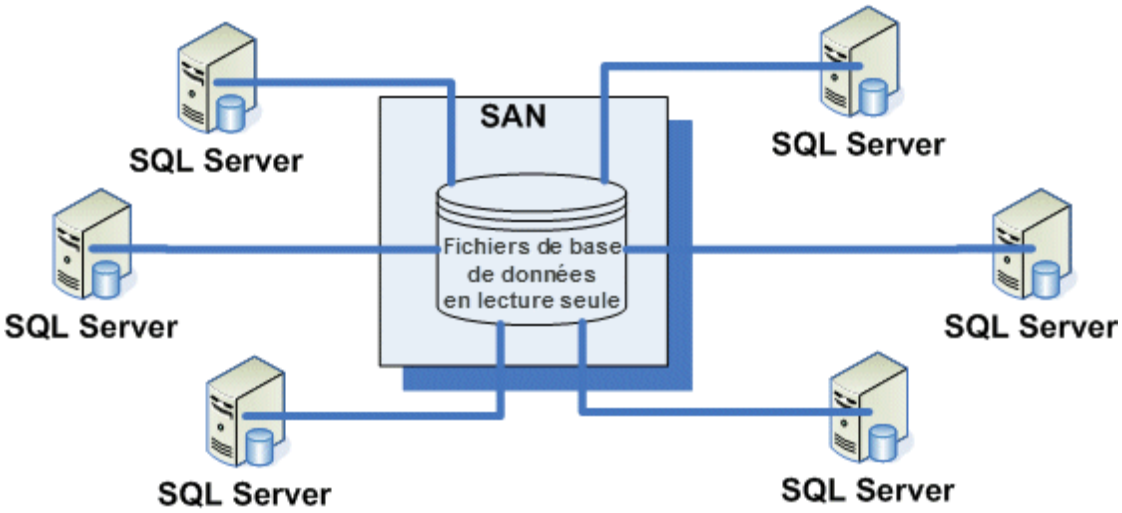

Figure 1. Base de données partagée évolutive

Bien entendu, une base de données qui n'évolue jamais offre un intérêt limité. Par conséquent, pour mettre à jour la base de données, toutes les instances de SQL Server détachent la base de données, une instance la lie en mode lecture-écriture et la base de données peut être actualisée avec des données à jour. Cette opération peut prendre un certain temps si elle implique la modification d'un grand nombre de données. Si le SAN possède suffisamment d'espace libre, il peut être intéressant de gérer deux bases de données de manière à mettre à jour l'une pendant que l'autre est utilisée. Pour plus d'informations, reportez-vous aux manuels en ligne sur SQL Server et aux articles indiqués à la section « Références ».

La limite de huit moteurs attachés à une seule base de données n'est pas d'ordre technique. Elle a été déterminée suite à la réalisation de tests. Les prochaines versions pourront vraisemblablement prendre en charge plus de huit moteurs. Puisque l'on n'écrit jamais dans la base de données et que le maximum de données possible est mis en cache, le taux réel d'E/S disque est relativement bas, même avec huit moteurs de base de données assumant une lourde charge.

Nombre d'applications de base de données les plus exigeantes fonctionnent parfaitement en lecture seule. La charge de travail d'un entrepôt de données consiste en un nombre relativement modéré de requêtes vastes et complexes concernant des données historiques raisonnablement statiques. La fréquence de mise à jour de la plupart des entrepôts de données est quotidienne, voire hebdomadaire. Ainsi, il est simple d'utiliser l'entrepôt en

lecture seule lorsqu'il fonctionne. Une base de données partagée évolutive permet également de résoudre un problème courant propre aux entrepôts de données : un utilisateur écrit une requête qui monopolise toutes les ressources de la base de données durant des heures. Lorsque cette requête s'exécute sur l'un des moteurs de base de données, les autres moteurs restent disponibles pour traiter les requêtes, ce qui minimise l'impact des « requêtes infernales ».

Les bases de données partagées évolutives ne sont utiles que si la fréquence de mise à jour est très faible, car la base de données ne peut pas être actualisée tant qu'elle est partagée. Les bases de données partagées évolutives ne nécessitent aucun changement au niveau de l'application, sous réserve que celle-ci ne tente pas de mettre à jour la base de données. La base de données est montée en charge horizontalement sans modification. Par conséquent, les facteurs de partitionnement et de couplage n'influencent pas la décision d'utiliser les bases de données partagées évolutives. En résumé, les bases de données partagées évolutives sont utiles dans des applications type entrepôts de données, mini-Data Warehouses et bases de données de création de rapports où la fréquence des mises à jour de données peut être limitée à des changements périodiques par lots. Si ce schéma de mise à jour est acceptable pour l'application, les bases de données partagées évolutives constituent la méthode de montée en charge horizontale privilégiée, car elles sont faciles à mettre en œuvre et nécessitent peu de changements au niveau de l'application.

Réplication d'égal à égal

Pour la montée en charge horizontale des données devant être mises à jour à une fréquence relativement modérée, on obtiendra souvent les meilleurs résultats par la réplication. Au lieu que plusieurs moteurs de base de données accèdent à une même copie de la base de données, celle-ci est disponible en plusieurs exemplaires pour que chaque moteur tienne à jour sa propre copie. Cette option offre à la fois la montée en charge horizontale et la mise à jour des données. La réplication permet de propager les changements vers toutes les copies des données. La Figure 2 présente l'utilisation de la réplication d'égal à égal comme solution de montée en charge horizontale.

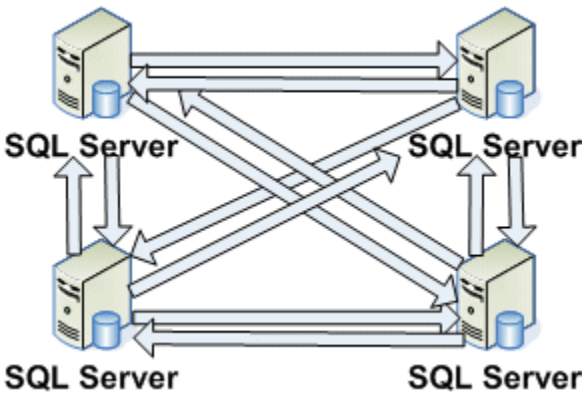

Figure 2. Réplication d'égal à égal

La réplication d'égal à égal dans SQL Server propage les changements apportés à une copie des données vers toutes les autres copies. Cette opération ne résout pas les conflits.

Elle n'est donc préconisée que dans les configurations où une seule copie d'un élément de données est mise à jour. Par exemple, si la réplication d'égal à égal est choisie pour gérer l'inventaire d'une chaîne de magasins d'alimentation, seul le point de vente détenteur d'une entrée d'inventaire serait habilité à mettre à jour cette entrée. Dans ce cas, tous les magasins pourraient consulter l'inventaire des autres, mais ne pourraient modifier que leur propre stock.

Les règles autorisant uniquement le détenteur d'un élément de données à le modifier sont appelées « gérance des données ». La gérance est une méthode extrêmement efficace pour éviter les conflits de mise à jour des données. Dans les cas où la gérance des données n'est pas envisageable, une réplication de fusion peut permettre de gérer les conflits. Cette méthode implique une surcharge plus importante que la réplication d'égal à égal, car elle doit gérer des conflits. On préfèrera donc la réplication d'égal à égal dans les cas où les conflits peuvent être évités. La réplication d'égal à égal doit être configurée à partir de chaque copie de la base de données vers chaque autre copie, de sorte que la gestion de cette structure peut s'avérer fastidieuse lorsque de nombreuses bases de données sont impliquées. La solution la plus simple et la plus efficace consiste à utiliser une seule copie principale avec réplication transactionnelle pour maintenir les autres copies à jour, sous réserve que les mises à jour de la base de données puissent être limitées à une seule copie de la base de données.

La réplication est une solution de montée en charge horizontale satisfaisante pour les données dont la fréquence de mise à jour est modérée. La gérance des données peut éliminer les conflits de mise à jour et permettre de recourir à une réplication d'égal à égal. Si besoin, la réplication peut être appliquée à toutes les données d'une même base de données. Elle est donc utile lorsque le partitionnement des données est compliqué ou que le degré de couplage est important. Dans la plupart des cas, elle ne nécessite pas de changement au niveau de l'application. Elle peut donc être utilisée pour faciliter la montée en charge horizontale des applications existantes. La réplication peut donner de bons résultats avec des tables uniques, voire des portions de tables. Elle est donc intéressante pour une montée en charge horizontale de parties de données d'une application. Par exemple, des données de référence (catalogues ou tarifs, par exemple) peuvent être répliquées pour optimiser la montée en charge, même si les données de ressources ne bénéficient pas de cette montée en charge. La réplication est également utile en association avec d'autres solutions de montée en charge horizontale. Si les données d'activité sont montées en charge horizontalement par répartition dans différentes bases de données, il peut être intéressant de répliquer les données de référence utilisées par les données d'activité vers toutes les bases de données d'activité. En général, la réplication est l'une des solutions de montée en charge horizontale les plus simples et les plus largement applicables dans SQL Server.

Reportez-vous à la page sur la topologie de réplication d'égal à égal pour effectuer des tâches courantes de configuration (ajout et suppression de nœuds, ajout de connexions entre des nœuds existants). Cet outil d'interface graphique convivial et intuitif facilite la création et la maintenance de la topologie de réplication d'égal à égal.

La réplication d'égal à égal dans SQL Server offre notamment la possibilité de repérer les conflits au sein d'une topologie d'égal à égal. Cette option permet d'éviter les problèmes

causés par des conflits non détectés, notamment des incohérences dans le comportement d'une application ou la perte de mises à jour. En activant cette option, un changement conflictuel est traité par défaut comme une erreur critique qui met en échec l'agent de distribution.

Serveurs liés et requêtes distribuées

SQL Server est capable d'interroger des objets de bases de données distantes comme s'il s'agissait de bases locales. Ainsi, une base de données montée en charge horizontalement peut se présenter à une application sous l'aspect d'une seule grande base de données. Le changement majeur pour les requêtes SQL réside dans le nom des tables qui inclura le nom du serveur lié sur lequel les données sont hébergées. Les serveurs liés constituent une option intéressante de montée en charge horizontale lorsqu'il est difficile de modifier les applications. Dans SQL Server, il est possible d'utiliser des synonymes pour soumettre un nom en quatre parties incluant le nom du serveur comme un nom unique. Ainsi, les requêtes qui renvoyaient à une table locale peuvent désormais renvoyer à une table distante sans modifier la requête. La Figure 3 présente l'utilisation de serveurs liés comme solution de montée en charge horizontale.

Figure 3. Serveurs liés

Les serveurs liés sont particulièrement utiles dans un environnement où les données peuvent être subdivisées par secteur fonctionnel parmi des bases de données faisant très peu appel au couplage. Les contraintes d'intégrité référentielle ne sont pas compatibles avec les tables distantes et les relations entre données locales et distantes doivent donc être minimisées. Les requêtes distantes sont considérablement plus coûteuses que les requêtes locales. Les jonctions entre tables locales et distantes peuvent elles aussi coûter très cher et les mises à jour des tables distantes nécessitent des transactions distribuées. Par conséquent, la montée en charge horizontale d'une base de données avec des serveurs liés nécessite une conception minutieuse de la base de données de manière à minimiser l'accès aux données distantes. Si les applications utilisent des îlots de données présentant peu de couplages et si les requêtes envoyées à ces îlots sont relativement peu nombreuses, il est possible de répartir les îlots de données entre des bases de données pour optimiser la montée en charge horizontale. S'il existe des « points chauds » où plusieurs applications utilisant différents îlots de données exploitent certaines tables de manière intensive, vous pouvez recourir à la réplication pour répondre au mieux aux requêtes impliquant ces tables locales.

Les bases de données montées en charge horizontalement doivent également être conçues pour que l'essentiel des données utilisées par une application soit stocké dans une même base de données. Par exemple, si plusieurs applications utilisent des données relatives aux clients et aux commandes dans la plupart de leurs requêtes, la séparation des données

clients et commandes dans des bases de données distinctes n'aura pas grand intérêt (même si les données sont librement couplées). En effet, quelle que soit la base de données ouverte par une application, elle accède en permanence aux données à distance. Toutefois, si certaines applications accèdent aux données clients de manière quasi exclusive tandis que d'autres se consacrent entièrement ou presque aux données de commandes, ce type de séparation peut s'avérer efficace. Là encore, si quelques requêtes font une utilisation intensive des deux bases de données, la réplication peut permettre de réduire le trafic réseau. Par exemple, si des commandes requièrent un numéro de client valide et si chaque commande inclut un nom de client, les colonnes Numéro et Nom du client de la base de données client peuvent être répliquées dans la base de données des commandes.

Tout comme la réplication, les serveurs liés sont de précieux outils dans certaines autres solutions de montée en charge horizontale. Dans toute base de données distribuée, un certain nombre de requêtes doit franchir les limites des bases de données. Les serveurs liés offrent un moyen simple de prendre en charge ces requêtes.

Les serveurs liés ont également leur utilité en cas de séparation de données par type. Par exemple, si toutes les informations historiques de plus de 60 jours sont transférées vers d'autres bases de données, la mise en œuvre d'un serveur lié peut permettre à l'application de gérer un petit nombre de requêtes qui accèdent aux données de référence historiques comme si elles étaient stockées localement. Grâce à cette stratégie, le traitement des données de référence est efficacement transféré vers un matériel dédié, sans affecter les applications qui les exploitent. Les applications qui accèdent fréquemment aux données historiques peuvent s'exécuter sur les systèmes historiques et atteindre les données actives uniquement lorsque c'est nécessaire.

La fréquence de mise à jour a un impact modéré sur une solution de montée en charge horizontale par serveur lié, sous réserve que les mises à jour ne couvrent pas les bases de données. Par conséquent, une validation en deux phases n'est pas nécessaire. Ce qui signifie qu'une solution de serveur lié est efficace pour certaines applications de traitement des transactions en ligne (OLTP). Par ailleurs, les serveurs liés nécessitent peu ou pas de changement au niveau applicatif. C'est pourquoi, ils conviennent tout à fait aux applications existantes. Le partitionnement des données par valeur clé n'est généralement pas utilisé dans les mises en œuvre de serveurs liés. Il n'entre donc pas en ligne de compte. Le principal facteur pour déterminer si la montée en charge horizontale par serveur lié constitue une solution appropriée est le couplage des données. Si le degré de couplage est élevé, la surcharge engendrée par les requêtes distribuées annulera les avantages de la montée en charge en termes de performances. Une bonne compréhension des relations entre les données et de l'utilisation des données d'application est essentielle pour savoir si les serveurs liés représentent une solution viable.

Vues partitionnées distribuées

Des vues partitionnées distribuées (DPV) ont été intégrées à SQL Server dans l'objectif précis d'assurer la montée en charge horizontale transparente des données partitionnées. Les données d'une table sont partitionnées entre les tables de plusieurs bases de données distribuées, en fonction d'une clé de partitionnement. Par exemple, une table client peut être partitionnée en plusieurs plages de numéros clients (1 à 10 000 dans une base de données,

10 001 à 20 000 dans une deuxième, 20 001 à 30 000 dans une troisième, et ainsi de suite). Des contraintes de vérification indiquent à SQL Server dans quelle base de données se trouvent les différents clients. Une requête qui accède à un numéro de client ne sera exécutée que dans la base de données qui contient le numéro de client souhaité. Les requêtes exemptes de numéro de client doivent être exécutées dans toutes les partitions. Par exemple, si vous souhaitez consulter des informations sur George Bush sans connaître son numéro de client, SQL Server doit effectuer une recherche dans toutes les bases de données contenant une partition de la table des clients. De même, une jonction entre deux tables sur le numéro de client peut s'exécuter parallèlement dans chaque base de données, les résultats étant renvoyés à la base d'où émane la requête.

À l'instar de la plupart des solutions de montée en charge horizontale, le recours aux DPV implique un nombre limité de requêtes nécessitant un transfert de données entre bases de données. Si toutes les données des bases de données DPV sont partitionnées à l'aide de la même clé de partitionnement, toutes les jonctions correspondant à cette clé peuvent s'exécuter en local. Si vous partitionnez une base de données de saisies de commandes de telle sorte que toutes les tables soient partitionnées sur la clé client, et si les requêtes de la base de données contiennent une clé client, toutes les requêtes peuvent être satisfaites dans la base de données qui contient cette clé. Bien entendu, si vous exécutez la requête depuis une base de données qui ne contient pas cette clé, la requête sera distribuée. En pratique, il n'est pas possible de partitionner toutes les données sur une même clé. Par conséquent, certaines tables doivent être partitionnées sur différentes clés et de nombreuses tables ne sont pas partitionnées dans la plupart des mises en œuvre. Ainsi, de nombreuses requêtes doivent accéder à plusieurs bases de données. La réussite de la montée en charge horizontale dépend donc de la composition relative de requêtes distribuées et locales. Une jonction entre une table partitionnée et une table non partitionnée peut s'avérer coûteuse du fait que les données non partitionnées doivent être envoyées à tous les serveurs hébergeant les partitions. Une proportion élevée de requêtes locales implique généralement une conception minutieuse du schéma de partitionnement, mais aussi la modification de l'application en vue de tirer pleinement parti du partitionnement. Ainsi, même si les DPV ont été conçues pour assurer une montée en charge horizontale transparente, dans bien des cas, la modification des applications reste nécessaire. Trouver un schéma de partitionnement adapté à plusieurs applications différentes n'est pas chose facile. C'est pourquoi, les DPV sont généralement plus intéressants lorsqu'un nombre limité d'applications associées exploite la base de données partitionnée. Pour limiter le nombre d'applications qui accèdent à un modèle DPV, il suffit de séparer les données par application et de les partitionner par valeur clé. En procédant ainsi, il est probable que seules quelques bases de données soient suffisamment volumineuses pour bénéficier des DPV et qu'il en résulte une combinaison de schémas partitionnés et non partitionnés.

Les DPV sont particulièrement performantes dans les applications aux mises à jour fréquentes, car la plupart des transactions d'actualisation affectent un petit nombre de lignes et peuvent donc s'exécuter dans une base de données unique. Les DPV représentent ainsi la solution de montée en charge horizontale la plus efficace en cas de mises à jour fréquentes. En théorie, une mise en œuvre de DPV ne devrait nécessiter aucun changement d'application, car les montées en charge horizontales sont gérées par SQL Server et transparentes au niveau de l'application. Dans la réalité, certains changements peuvent être requis dans l'application pour tirer pleinement parti de la montée en charge. Le facteur qui

détermine le choix de mettre en œuvre des DPV est la capacité de partitionnement des données. En l'absence de clés de partitionnement efficaces, capables de partitionner les données de manière homogène et de minimiser les requêtes entre les partitions, les DPV ne sont d'aucun secours. Choisir les clés de partitionnement adaptées requiert un important travail d'analyse et de planification en amont. Le couplage des données n'a que peu d'impact sur une vue partitionnée, mais si les bases de données contiennent de nombreuses tables non partitionnées, un taux de couplage élevé peut entraver la bonne distribution des données.

Les applications OLTP sont souvent les meilleures candidates aux vues partitionnées distribuées car elles allient d'importants volumes de données à des fréquences de mise à jour élevées. D'une manière générale, une vue partitionnée demande un effort de gestion beaucoup plus conséquent que si les données se trouvaient dans une seule table. Les opérations de sauvegarde et de restauration doivent pouvoir synchroniser toutes les partitions d'une table pour restaurer un état cohérent, par exemple. Cet effort de gestion supplémentaire, associé à celui relativement important de la mise en place d'une solution DPV, explique principalement le faible nombre d'applications exploitant les DPV comme solution de montée en charge horizontale.

Routage dépendant des données

Avec les vues partitionnées distribuées, SQL Server comprend comment les données sont partitionnées et détermine où les chercher. Avec le routage dépendant des données (DDR), les informations sont partitionnées parmi les bases de données. L'application ou certains services middleware se chargent ensuite d'acheminer les requêtes vers la bonne base de données. Le DDR est donc beaucoup moins transparent au niveau applicatif. Toutefois, si l'application intervient dans le choix du point d'exécution d'une requête, plus elle dispose d'informations sur les données, plus sa décision pourra être judicieuse. Le DDR peut également prendre en charge des options non gérées directement par SQL Server actuellement. Par exemple, si plusieurs copies des mêmes données sont disponibles, le DDR peut distribuer des commandes SELECT entre ces copies, tout en dirigeant les mises à jour vers la copie principale. La Figure 4 présente l'utilisation du routage dépendant des données comme solution de montée en charge horizontale.

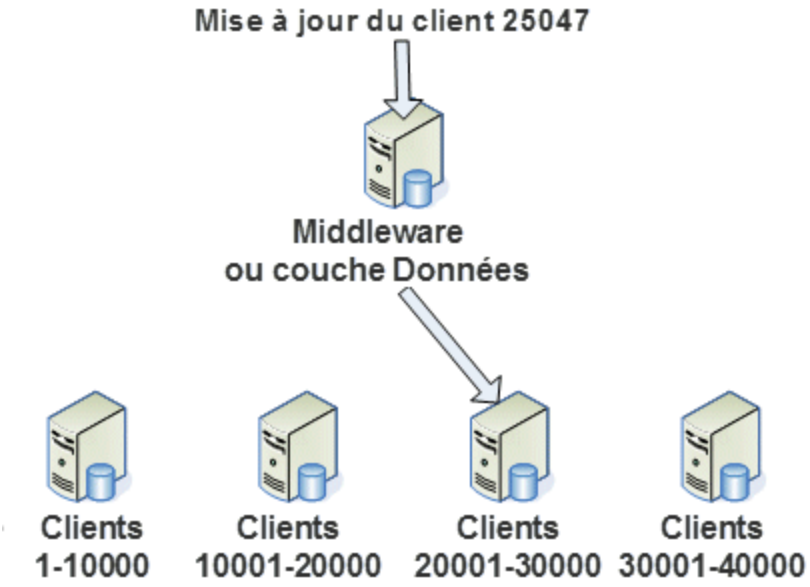

Figure 4. Routage dépendant des données

Beaucoup de grandes bases de données qui gèrent d'importants volumes de transactions utilisent des variantes de la montée en charge horizontale par DDR pour répartir le traitement parmi des centaines, voire des milliers de serveurs de bases de données. Face à une telle quantité de serveurs, la plupart des requêtes doivent être dirigées vers un seul serveur. Le modèle de données doit donc être conçu pour que toutes les données nécessaires pour une requête ou une mise à jour soient hébergées sur le même serveur. Par exemple, un grand distributeur en ligne peut partitionner ses bases de données en fonction de l'identifiant client. Ainsi, toutes les informations concernant un client, toutes les commandes qu'il a passées, le statut de ses factures, ses informations de carte bancaire, etc. se trouvent dans la même base données. Ce sont les données de ce client précis. On parle souvent dans ce cas d'entité client. Le partitionnement de données entre des entités (groupes de données indépendants auquel un seul identifiant peut faire référence) est la méthode la plus utilisée pour concevoir des bases de données très volumineuses. Chaque moteur de base de données gère des requêtes pour les entités clients de sa propre base. Lorsque le nombre d'entités devient trop important, un nouveau serveur est ajouté et les entités sont redistribuées.

Une fois que toutes les données clients sont partitionnées en entités identifiées par un ID client, reste à déterminer comment l'application sait quel ID client utiliser. La plupart du temps, c'est assez simple. Si le client est censé passer des commandes, on peut supposer qu'une connexion a été créée et que l'ID client est dans un enregistrement de connexion (sur le serveur Web ou dans un cookie, par exemple). Si aucune procédure de connexion n'existe, il suffit de mettre en œuvre une table mappant l'identité des clients (nom, SID Windows, etc.) vers un ID client. Dernière pièce du puzzle : une table chargée de mapper les ID clients vers le serveur de base de données sur lequel l'entité client est enregistrée. Une fois ce mécanisme en place, une couche d'accès aux données (dans l'application ou dans une application intermédiaire) peut soumettre une API permettant à l'application d'accéder aux données stockées dans une entité client donnée. Les requêtes sont

acheminées en fonction des données transmises par la requête, d'où le nom de « routage dépendant des données ».

À ce stade, il est logique de se demander comment lancer une recherche dans plusieurs centaines de bases de données pour obtenir des données de résumé ou trouver toutes les commandes. La réponse est simple : « Ne le faites pas. » Dans le cadre de la conception d'une application DDR, vous devez mettre en place les installations nécessaires pour gérer ces requêtes. Une approche type consiste à répliquer des données de résumé de chaque commande dans une base de données de commandes en les associant à un ID client, de telle sorte que la plupart des requêtes de commande puissent accéder à ces données de résumé. Si une application requiert les détails d'une commande, elle peut utiliser l'ID client des données de résumé pour retrouver la commande. Les données de résumé sont aussi couramment chargées dans un entrepôt ou dans des bases de données de création de rapports pour différentes utilisations (génération de rapports, exploration de données, analyse de données, etc.).

Bien qu'il ne soit pas très courant d'avoir recours au DDR pour réaliser des montées en charge horizontales de milliers de serveurs de bases de données, l'utilisation de ces mêmes principes pour monter en charge des dizaines de serveurs de bases de données peut s'avérer viable pour de nombreuses applications. La relocalisation des données, la gestion de la réplication ou l'extraction des données de résumé rendent cette solution relativement complexe à gérer. Mais pour l'essentiel, ces tâches sont répétitives et peuvent être automatisées.

La solution DDR a été pensée pour des volumes de transactions conséquents. Elle s'impose donc tout naturellement pour les applications avec une fréquence de mise à jour élevée. Le DDR a besoin d'une couche de données qui puisse localiser les entités de données et y accéder depuis les bases de données où elles sont stockées. Ainsi, le DDR est la solution la plus adaptée pour une nouvelle application. Le recours à une application intermédiaire pour gérer le routage des données réduira le nombre de changements apportés au code de l'application. Mais l'adaptation d'une application existante au DDR nécessite des modifications importantes. Les exigences liées à la capacité de partitionnement des données sont extrêmement élevées. En effet, cette solution ne fonctionne que si les données sont partitionnées et qu'une clé de partitionnement permette de repérer toutes les entités de données. Un faible degré de couplage des données peut également être requis, car il est généralement impossible de partitionner la totalité de la base de données sur une seule clé. Dans notre exemple de saisie de commande, le partitionnement de l'inventaire par ID client n'aurait aucun intérêt. Les données d'inventaire devraient être isolées dans leur propre ensemble de serveurs de bases de données, voire partitionnées par numéro de catalogue. En général, le DDR est l'une des méthodes les plus efficaces pour monter en charge horizontalement une base de données, mais c'est aussi l'une des plus difficiles à mettre en œuvre. Elle est souvent plus appropriée dans les cas de reconception ou de modification d'applications. Le DDR convient également à la montée en charge horizontale partielle d'une application. Par exemple, vous pouvez appliquer le DDR à votre base de données de commandes et utiliser une autre stratégie de montée en charge horizontale pour le reste des données. Vous pouvez aussi opter pour une montée en charge verticale des autres données si l'extraction des données de commande laisse un volume de données suffisamment modéré pour être géré ainsi.

Fédération SQL Azure

Face aux besoins hautement évolutifs du cloud computing, Microsoft a instauré un principe de fédération dans Microsoft SQL Azure™ pour augmenter la capacité de montée en charge horizontale, parer aux problèmes de répartition et de redistribution des données, et répondre à la nécessité d'un routage de connexion performant. La fédération permet aux applications d'évoluer de plusieurs dizaines à plusieurs centaines de bases de données SQL. Elle facilite également le repartitionnement des données sans interruption d'activité. Pour répondre aux exigences d'architecture mutualisée, la fédération fournit des opérations de repartitionnement qui facilitent la gestion du positionnement et du repositionnement des locataires, sans que l'application ne soit interrompue. Parmi les applications de base de données pouvant bénéficier de la fédération, on retrouve les éditeurs de logiciels qui mettent en œuvre des solutions SaaS mutualisées, les solutions de base de données Web à grande échelle visant à faire face aux pics de demandes, aux rafales, aux mines ou aux nouveaux flux d'utilisateurs, et les applications NoSQL.

Conclusion

Retenons de cette discussion sur la montée en charge horizontale de SQL Server que les applications contiennent différents types de données. Une solution efficace de montée en charge peut intégrer des approches différentes pour chaque type de données. Les données de référence peuvent être répliquées et mises en cache à plusieurs emplacements ; les données historiques peuvent être exposées via des requêtes distribuées dans un système de stockage économique de grande capacité ; les données d'activité peuvent être partitionnées sur plusieurs serveurs ; enfin, les données de ressources peuvent être réparties par application.

Le choix d'une solution de montée en charge horizontale repose sur un certain nombre de facteurs. Le Tableau 1 récapitule l'importance de ces facteurs pour chaque solution.

Tableau 1. Facteurs influençant le choix des solutions de montée en charge horizontale

	Fréquence de mise à jour	Aptitude à modifier l'application	Capacité de partitionnement des données	Couplage des données
Bases de données partagées évolutives	Lecture seule.	Peu, voire pas de changement requis.	Sans importance.	Sans importance.
Réplication d'égal à égal	Lecture essentiellement, pas de conflits.	Peu, voire pas de changement requis.	Sans importance.	Sans importance.
Serveurs liés	Mises à jour minimisées entre les bases de données.	Changements mineurs.	Généralement sans importance.	Faible taux de couplage essentiel.
Vues partitionnées distribuées	Mises à jour fréquentes prises en charge.	Certains changements peuvent être requis.	Très important.	Impact minime.
Routage dépendant des données	Mises à jour fréquentes prises en charge.	Changements importants possibles.	Très important.	Un faible taux de couplage peut être utile pour certaines applications.

Rappelons que certaines architectures de montée en charge horizontale peuvent incorporer plusieurs solutions. La réplication et les serveurs liés font généralement partie de ce type d'architecture.

Avec une bonne compréhension des données, des besoins et des contraintes applicatives, il est possible de développer une solution SQL Server efficace et capable de prendre en charge tout type de montée en charge horizontale. Même si la montée en charge horizontale n'est pas une exigence initiale pour une application, considérer cette évolution dès la conception peut se révéler payant par la suite, lorsque l'application et votre activité se développeront.

Si vous avez besoin d'une solution de montée en charge horizontale d'envergure pour des applications de bases de données en mode cloud, la fédération SQL Azure introduit le concept d'informatique souple qui offre des possibilités de montée en charge horizontale vers des centaines de nœuds mais aussi des possibilités de régression.

Références :

Site Web SQL Server : http://www.microsoft.com/sqlserver/en/us/default.aspx

Base de données partagée évolutive

- Bases de données partagées évolutives : http://msdn.microsoft.com/en-us/library/ms345584.aspx

Réplication d'égal à égal

- Réplication transactionnelle d'égal à égal : http://msdn.microsoft.com/en-us/library/ms151196(v=sql.110).aspx
- Configuration de la topologie de réplication transactionnelle d'égal à égal : http://msdn.microsoft.com/en-us/library/ms183664.aspx
- Détection de conflits dans la réplication transactionnelle d'égal à égal : http://msdn.microsoft.com/en-us/library/bb934199(v=sql.110).aspx

Serveurs liés

- Liaison de serveurs : http://msdn.microsoft.com/en-us/library/ms188279.aspx

Vues partitionnées distribuées

- Mise en œuvre de serveurs de bases de données fédérés : http://msdn.microsoft.com/en-us/library/ms191185.aspx
- Résolution des vues partitionnées distribuées : http://technet.microsoft.com/en-us/library/ms187836.aspx
- Modification de données dans des vues partitionnées distribuées : http://msdn.microsoft.com/en-us/library/ms187067.aspx

Routage dépendant des données

- Montée en charge horizontale de SQL Server par le routage dépendant des données : http://technet.microsoft.com/en-us/library/cc966448.aspx

Fédération SQL Azure

- Fédération dans SQL Azure : http://msdn.microsoft.com/en-us/library/windowsazure/hh597452.aspx

Ce livre blanc vous a-t-il été utile ? N'hésitez pas à nous faire part de vos commentaires. Sur une échelle allant de 1 (mauvais) à 5 (excellent), quelle note attribueriez-vous à ce livre blanc et pourquoi ? Par exemple :

> Lui attribuez-vous une note élevée pour ses exemples utiles, ses captures d'écran pertinentes ou sa clarté rédactionnelle ?
> Lui attribuez-vous une mauvaise note à cause de ses exemples insuffisants, ses captures d'écran floues ou son style confus ?

Vos commentaires nous aideront à améliorer la qualité de nos livres blancs.

Envoyez vos commentaires.