Bonnes pratiques de migration d'un SGBDR vers Amazon DynamoDB

Utilisation de la puissance de NoSQL pour des charges de travail appropriées

Nathaniel Slater

Mars 2015

Table des matières

Résumé

À l'heure actuelle, les architectes et les développeurs de logiciels disposent d'une série de solutions pour le stockage et la persistance des données. Cela inclut non seulement des systèmes de gestion de base de données relationnelles (SGBDR) classiques, mais également des bases de données NoSQL, comme Amazon DynamoDB. Certaines charges de travail évolueront et seront plus rentables pour une exécution à l'aide d'une solution NoSQL. Ce document mettra en évidence les bonnes pratiques de migration de ces charges de travail d'un SGBDR vers DynamoDB. Nous expliquerons comment des bases de données NoSQL comme DynamoDB diffèrent d'un SGBDR classique et nous proposerons un cadre pour l'analyse, la modélisation de données et la migration de données d'un SGBDR vers DynamoDB.

Introduction

Pendant des dizaines d'années, le SGBDR était le choix pour le stockage et la persistance des données. Toute application orientée données, qu'il s'agisse d'un site Web d'e-commerce ou d'un système de génération de rapports de dépenses, utilisait presque toujours une base de données relationnelle pour récupérer et stocker les données requises par l'application. Les raisons sont nombreuses, notamment :

- Le SGBDR est une technologie évoluée et stable.
- Le langage de requête, SQL, est versatile et présente de nombreuses fonctions.
- Les serveurs qui exécutent un moteur de SGBDR font généralement partie des systèmes les plus stables et puissants de l'infrastructure informatique.
- Tous les principaux langages de programmation comportent une assistance pour les pilotes utilisés pour communiquer avec un SGBDR, ainsi qu'un grand ensemble d'outils pour simplifier le développement d'applications reposant sur une base de données.

Ces facteurs, et de nombreux autres, ont pris en charge cette période d'utilisation du SGBDR. Pour les architectes et les développeurs de logiciels, il n'existait simplement pas d'alternative raisonnable pour le stockage et la persistance des données, jusqu'à aujourd'hui.

Le développement d'applications Web à l'échelle d'Internet, comme l'e-commerce et les réseaux sociaux, l'explosion des appareils connectés comme des smartphones et des tablettes, et l'arrivée du Big Data ont entraîné de nouvelles charges de travail que les bases de données relationnelles classiques ne peuvent pas prendre en charge. En tant que système conçu pour le traitement des transactions, les propriétés fondamentales que tous les SGBDR doivent prendre en charge sont définies par l'acronyme ACID : atomicité, cohérence, isolation et durabilité. L'atomicité signifie « tout ou rien » : une transaction est exécutée totalement ou pas du tout. La cohérence signifie que l'exécution d'une transaction entraîne une transition d'état valide. Une fois la transaction validée, l'état des données qui en résultent doit être conforme aux limites imposées par le schéma de base de données. L'isolation nécessite que des transactions simultanées soient exécutées séparément les unes des autres. La propriété d'isolation garantit que si des transactions simultanées étaient exécutées en série, l'état final des données serait le même. La durabilité nécessite que l'état des données soit préservé lors de l'exécution d'une transaction. En cas de problème d'alimentation ou de défaillance système, la base de données doit pouvoir récupérer depuis le dernier état connu.

Ces propriétés ACID sont bénéfiques, mais la prise en charge des quatre propriétés nécessite une architecture qui présente des défis pour les charges de travail actuelles gourmandes en données. Par exemple, la cohérence nécessite un schéma bien défini et que toutes les données stockées dans une base de données soient conformes à ce schéma. C'est idéal pour des requêtes ponctuelles et des charges de travail impliquant une demande intensive en lecture. Pour une charge de travail composée presque entièrement d'écritures, comme l'enregistrement de l'état d'un lecteur dans une application de jeu, la mise en œuvre de ce schéma est onéreuse du point de vue du stockage et du calcul. Le développeur de jeux dispose de peu d'avantages : il force ces données dans des lignes et tableaux qui sont liés les uns aux autres via un ensemble bien défini de clés.

La cohérence nécessite également le blocage d'une partie des données jusqu'à ce que la transaction qui les modifie soit terminée, de manière à ce que les modifications soient immédiatement visibles. Pour une transaction bancaire qui débite un compte et en crédite un autre, c'est obligatoire. Ce type de transaction est à cohérence forte. D'un autre côté, pour une application de réseaux sociaux, il n'est pas exigé que tous les utilisateurs voient une mise à jour du flux de données au même moment précisément. Dans ce cas, la transaction est cohérente à terme. Il est bien plus important que l'application de réseaux sociaux évolue pour traiter potentiellement des millions d'utilisateurs simultanés même si ces utilisateurs constatent des changements de données à différents moments. Le dimensionnement d'un SGBDR pour traiter ce niveau de simultanéité tout en conservant une cohérence forte nécessite une mise à niveau vers du matériel plus puissant (et souvent propriétaire). C'est ce que l'on appelle le dimensionnement vers le haut ou vertical, et il s'accompagne généralement de frais très élevés. La manière plus rentable de faire évoluer une base de données de manière à prendre en charge ce niveau de simultanéité est d'ajouter des instances de serveur exécutées sur du matériel de base. C'est ce que l'on appelle le dimensionnement vers le bas ou horizontal, et il est généralement bien plus onéreux que le dimensionnement vertical.

Des bases de données NoSQL, comme Amazon DynamoDB, relèvent les défis de dimensionnement et de performances du SGBDR. Le terme « NoSQ » signifie simplement que la base de données ne suit pas le modèle relationnel pour lequel E.F Codd a opté dans son document de 1970 *A Relational Model of Data for Large Shared Data Banks*,[1] qui deviendrait la base du SGBDR moderne. Par conséquent, les bases de données NoSQL varient bien plus au niveau des fonctions et des fonctionnalités qu'un SGBDR classique. Il n'existe aucun langage de requête commun analogue à SQL, et la flexibilité des requêtes est généralement remplacée par des performances d'E/S élevées et une évolutivité horizontale. Les bases de données NoSQL n'intègrent pas la notion de schéma de la même manière qu'un SGBDR. Quelques bases de données peuvent stocker des données semi-structurées, comme JSON. D'autres peuvent stocker des valeurs connexes comme ensembles de colonnes. D'autres encore peuvent simplement stocker des paires clé-valeur.

[1] http://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf

Le résultat net est que des bases de données NoSQL échangent certaines capacités de requête et propriétés ACID d'un SGBDR pour un modèle de données bien plus flexible à dimensionnement horizontal. Avec ces caractéristiques, les bases de données NoSQL sont excellentes dans des cas où l'utilisation d'un SGBDR pour des charges de travail non relationnelles (comme l'exemple d'état de jeu susmentionné) entraîne une association de goulets d'étranglement de performances, de complexité opérationnelle et d'augmentation des coûts. DynamoDB propose des solutions à tous ces problèmes et est une plateforme idéale pour la migration de ces charges de travail d'un SGBDR.

Présentation d'Amazon DynamoDB

Amazon DynamoDB est un service de base de données NoSQL entièrement opéré exécuté dans le cloud AWS. La complexité de l'exécution d'une base de données NoSQL distribuée et très évolutive est gérée par le service lui-même, ce qui permet aux développeurs de logiciels de se concentrer sur la création d'applications plutôt que sur la gestion d'infrastructure. Les bases de données NoSQL sont conçues pour être évolutives, mais leur architecture est sophistiquée et il peut y avoir une surcharge de fonctionnement au niveau de l'exécution d'un grand cluster NoSQL. Au lieu de devoir devenir des experts en concepts de calcul distribués avancés, le développeur doit uniquement apprendre à connaître l'API simple de DynamoDB à l'aide du kit SDK pour le langage de programmation choisi.

Outre sa simplicité d'utilisation, DynamoDB est également économique. Avec DynamoDB, vous payez pour le stockage que vous utilisez et le débit des E/S que vous avez alloué. Il est conçu pour un dimensionnement élastique. Lorsque les exigences de stockage et de débit d'une application sont faibles, seule une petite partie des capacités doit être allouée au service DynamoDB. À mesure que le nombre d'utilisateurs d'une application et que le débit des E/S requis augmentent, des capacités supplémentaires peuvent être allouées à la volée. Cela permet à une application de se développer en toute transparence pour prendre en charge des millions d'utilisateurs qui font des milliers de demandes simultanées à la base de données par seconde.

Les tables constituent le composant fondamental pour l'organisation et le stockage de données dans DynamoDB. Une table est composée d'éléments. Un élément est composé d'une clé principale qui l'identifie de manière unique, et de paires clé-valeur appelées attributs. Alors qu'un élément est similaire à une ligne d'une table de SGBDR, tous les éléments de la même table DynamoDB ne doivent pas partager le même ensemble d'attributs de manière à ce que toutes les lignes d'une table relationnelle partagent les mêmes colonnes. La Figure 1 montre la structure d'une table DynamoDB et des éléments qu'elle contient. Il n'existe pas de concept de colonne dans une table DynamoDB. Chaque élément de la table peut être exprimé comme un tuple comprenant un nombre arbitraire d'éléments, jusqu'à une taille maximale de 400 K. Ce modèle de données est bien adapté au stockage des données dans des formats fréquemment utilisés pour la sérialisation d'objet et la messagerie dans des systèmes distribués. Comme nous le verrons dans la section suivante, les charges de travail qui impliquent ce type de données sont de bons candidats à la migration vers DynamoDB.

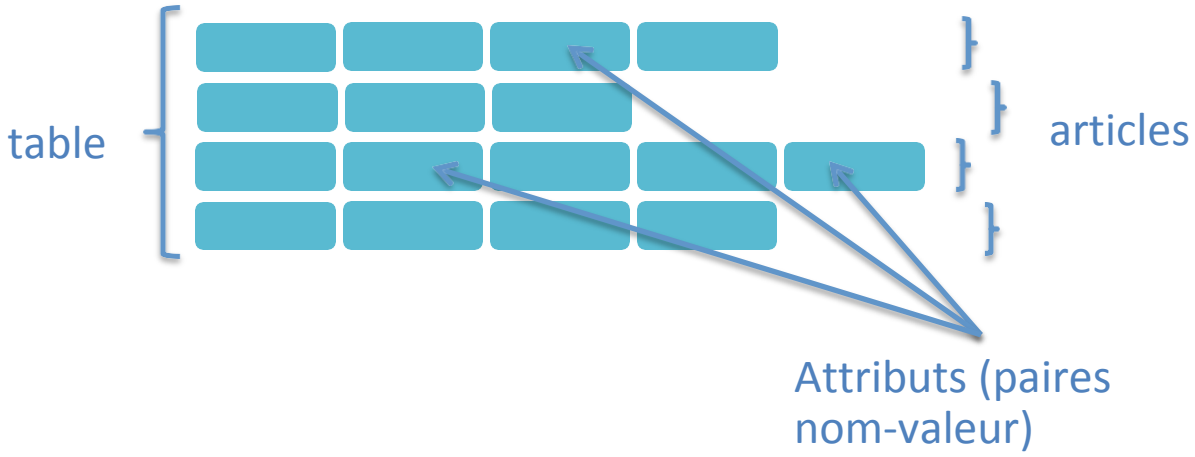

Figure 1 : Structure de table DynamoDB

Les tables et éléments sont créés, mis à jour et supprimés via l'API DynamoDB. Il n'existe pas de concept de langage DML standard comme pour les bases de données relationnelles. La manipulation des données dans DynamoDB est effectuée par programmation via un code orienté objet. Il est possible d'interroger des données dans une table DynamoDB, mais cela est également réalisé par programmation grâce à l'API. Étant donné qu'il n'existe aucun langage de requête générique comme SQL, il est important de bien comprendre les modèles d'accès aux données de votre application pour utiliser au mieux DynamoDB.

Charges de travail adaptées

DynamoDB est une base de données NoSQL, ce qui signifie qu'elle offrira de meilleures performances pour des charges de travail impliquant des données non relationnelles. Certains des cas d'utilisation les plus fréquents pour des charges de travail non relationnelles sont les suivants :

- Ingénierie publicitaire
  - Capturer l'état des cookies du navigateur
- Applications mobiles
  - Stocker des données d'application et l'état de la session
- Applications de jeu
  - Stocker des préférences d'utilisateur et l'état de l'application
  - Stocker l'état des jeux des joueurs
- Applications de vote client
  - Concours de téléréalité, publicités pendant le Superbowl

- Sites Web à grande échelle
  - État de la session
  - Données utilisateur utilisées à des fins de personnalisation
  - Contrôle d'accès
- Surveillance des applications
  - Stocker des données du journal d'application et d'événement
  - Données JSON
- Internet des objets
  - Intégration de données de capteur et de journaux

Tous ces cas d'utilisation bénéficient d'une série de fonctions qui rendent les bases de données NoSQL si puissantes. Les applications d'ingénierie publicitaire nécessitent généralement une latence très faible qui convient parfaitement aux performances de lecture et d'écriture faibles à un seul chiffre pour les millisecondes de DynamoDB. Des applications mobiles et de vote client ont souvent des millions d'utilisateurs et doivent gérer des milliers de demandes par seconde. DynamoDB peut procéder au dimensionnement horizontal pour respecter cette charge. Enfin, des solutions de surveillance d'application intègrent généralement des centaines de milliers de points de données par minute, et le modèle de données dépourvu de schéma DynamoDB, des performances d'E/S élevées et la prise en charge d'un type de données JSON natif idéal pour ces types d'applications.

Une autre caractéristique importante à prendre en compte pour déterminer si une charge de travail est adaptée pour une base de données NoSQL comme DynamoDB est de savoir si elle nécessite un dimensionnement horizontal. Une application mobile peut avoir des millions d'utilisateurs, mais chaque installation de l'application ne lira et n'écrira des données de session que pour un seul utilisateur. Cela signifie que les données de la session de l'utilisateur dans la table DynamoDB peuvent être réparties sur plusieurs partitions de stockage. Une lecture ou une écriture de données pour un utilisateur donné sera limitée à une seule partition. Cela permet le dimensionnement horizontal de la table DynamoDB : à mesure que des utilisateurs sont ajoutés, des partitions sont créées. Tant que les demandes de lecture et d'écriture de ces données sont réparties uniformément sur des partitions, DynamoDB pourra gérer un très grand accès aux données simultanées. Ce type de dimensionnement horizontal est difficile à réaliser avec un SGBDR sans utiliser le partitionnement, ce qui peut sensiblement rendre plus complexe la couche d'accès aux données d'une application. Lorsque les données dans un SGBDR sont partitionnées, elles sont réparties sur différentes instances de base de données. Cela nécessite de conserver un index des instances et la plage des données qu'elles contiennent. Pour lire et écrire des données, une application cliente doit savoir quelle partition contient la plage de données à écrire ou lire. Le partitionnement ajoute également une surcharge administrative et des coûts. Au lieu d'une seule instance de base de données, vous devez désormais en conserver plusieurs en état de fonctionner.

Il est également important d'évaluer l'exigence de cohérence des données d'une application pour déterminer si une charge de travail serait adaptée à DynamoDB. Il existe en effet deux modèles de cohérence pris en charge par DynamoDB : cohérence forte et à terme, la première nécessitant plus d'E/S provisionnées que la seconde. Cette flexibilité permet au développeur d'obtenir les meilleures performances possibles de la base de données, tout en pouvant assurer les exigences de cohérence de l'application. Si une application ne nécessite pas de lectures à cohérence forte, ce qui signifie que des mises à jour effectuées par un client ne doivent pas être visibles immédiatement pour les autres, l'utilisation d'un SGBDR qui forcera une cohérence forte pourra entraîner une taxation des performances sans bénéfice net pour l'application. La raison est que la cohérence forte implique généralement d'avoir à bloquer certaines parties de données qui peuvent entraîner des goulets d'étranglement de performances.

Charges de travail non adaptées

Toutes les charges de travail ne sont pas adaptées à une base de données NoSQL comme DynamoDB. Alors qu'en théorie on peut mettre en œuvre un modèle de relation d'entité classique à l'aide de tables et d'éléments DynamoDB, en pratique, leur utilisation serait fastidieuse. Des systèmes transactionnels qui nécessitent des relations bien définies entre des entités sont toujours mieux utilisés à l'aide d'un SGBDR classique. Certaines charges de travail non adaptées incluent :

- Requêtes ponctuelles
- OLAP
- Stockage sur BLOB

Étant donné que DynamoDB ne prend pas en charge un langage de requête standard comme SQL et qu'il n'y a pas de concept de jointure de table, la création de requêtes ponctuelles n'est pas aussi efficace qu'avec un SGBDR. L'exécution de ce type de requêtes avec DynamoDB est possible, mais nécessite l'utilisation d'Amazon EMR et de Hive. De même, les applications OLAP sont également difficiles à fournir, car le modèle de données dimensionnel utilisé pour le traitement analytique nécessite des tables de faits de jointure pour dimensionner des tables. Enfin, en raison de la limite de taille d'un élément DynamoDB, le stockage d'objets BLOB n'est souvent pas pratique. DynamoDB ne prend pas en charge un type de données binaire, mais ne convient pas pour le stockage de grands objets binaires, comme des images ou des documents. Toutefois, le stockage d'un pointeur dans la table DynamoDB sur un grand objet BLOB stocké dans Amazon S3 prend facilement en charge ce dernier cas d'utilisation.

Concepts clés

Comme décrit dans la section précédente, DynamoDB organise des données en tables composées d'éléments. Chaque élément d'une table DynamoDB peut définir un ensemble arbitraire d'attributs, mais tous les éléments de la table doivent définir une clé primaire qui identifie l'élément de manière unique. Cette clé doit contenir un attribut appelé « clé de hachage », voire un attribut appelé « clé de plage ». La Figure 2 montre la structure d'une table DynamoDB définissant une clé de hachage et de plage.

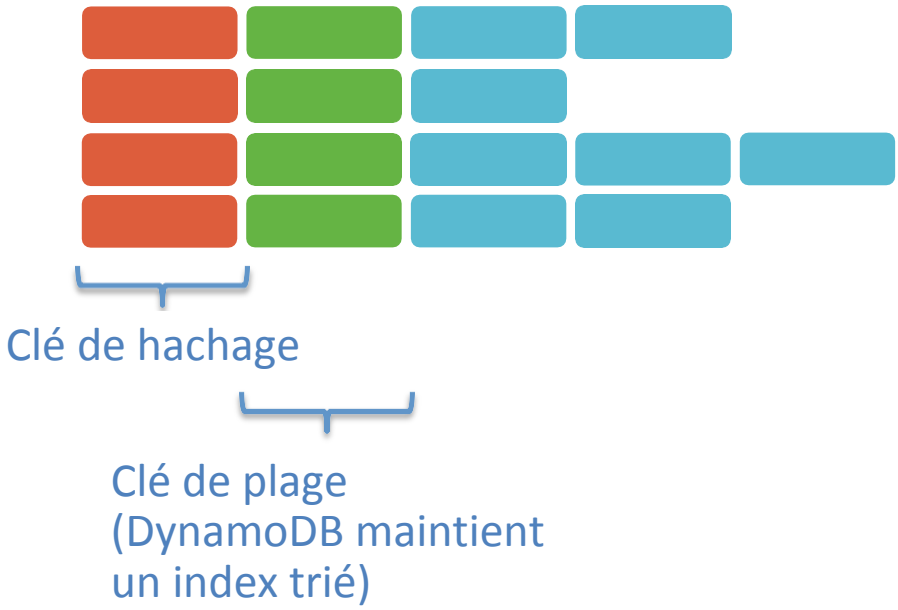

Figure 2 : Table DynamoDB avec clés de hachage et de plage

Si un élément peut être identifié de manière unique par une valeur d'attribut unique, cet attribut peut être utilisé comme clé de hachage. Dans d'autres cas, un élément peut être identifié de manière unique par deux valeurs. Dans ce cas, la clé primaire sera définie comme un composite de la clé de hachage et de la clé de plage. La figure 3 présente ce concept. Une table de SGBDR liée à des fichiers multimédias avec le codec utilisé pour les transcoder peut être modélisée comme une table unique dans DynamoDB à l'aide d'une clé primaire composée d'une clé de hachage et d'une clé de plage. Notez la manière dont les données sont dénormalisées dans la table DynamoDB. Il s'agit d'une pratique commune lors de la migration des données d'un SGBDR vers une base de données NoSQL qui sera présentée en détails plus loin dans ce document.

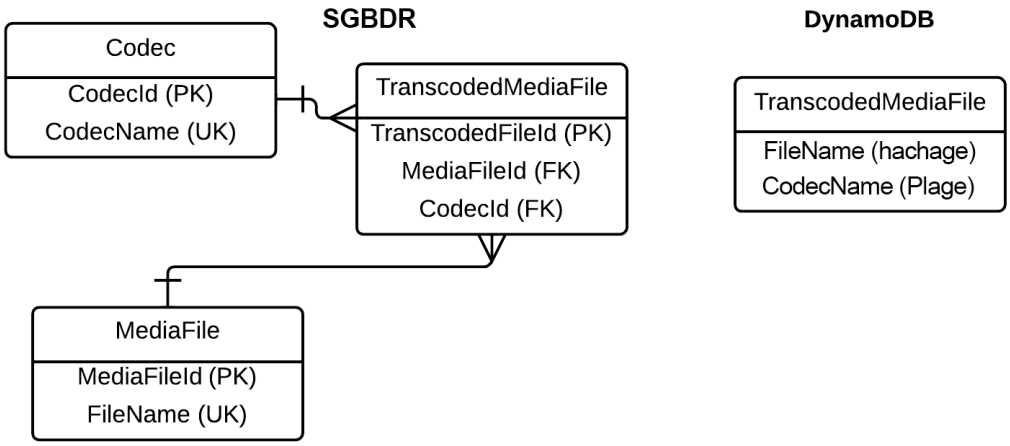

Figure 3 : Exemple de clés de hachage et de plage

La clé de hachage idéale contiendra un grand volume de données distinctes réparties de manière uniforme sur les éléments de la table. Un ID utilisateur est un bon exemple d'attribut qui a tendance à être réparti de manière uniforme sur différents éléments d'une table. Des attributs qui seraient modélisés comme des valeurs de recherche ou des énumérations dans un SGBDR ont tendance à donner des clés de hachage faibles. La raison est que certaines valeurs peuvent apparaître bien plus souvent que d'autres. Ces concepts sont illustrés à la Figure 4. Notez que le nombre de user_id est uniforme, alors que le nombre de status_code ne l'est pas. Si le status_code est utilisé comme clé de hachage dans une table DynamoDB, la valeur qui apparaît le plus souvent finira par être stockée sur la même partition, et cela signifie que la plupart des lectures et écritures concerneront cette seule partition. Il s'agit d'une partition à chaud, qui aura un impact négatif sur les performances.

```
select user_id, count(*) as total from user_preferences group by
user_id
```

user_id	total
8a9642f7-5155-4138-bb63-870cd45d7e19	1
31667c72-86c5-4afb-82a1-a988bfe34d49	1
693f8265-b0d2-40f1-add0-bbe2e8650c08	1

```
select status_code, count(*) as total from status_code sc, log l
where l.status_code_id = sc.status_code_id
```

status_code	total
400	125000
403	250
500	10000
505	2

Figure 4 : Répartition uniforme et non uniforme des valeurs de clé potentielles

Des éléments peuvent être extraits d'une table à l'aide de la clé primaire. Souvent, il est utile de pouvoir extraire des éléments à l'aide d'un ensemble de valeurs différent des clés de hachage et de plage. DynamoDB prend en charge ces opérations via des index secondaires locaux et globaux. Un index secondaire local utilise la même clé de hachage que définie sur la table, mais un autre attribut comme clé de plage. La Figure 5 montre comment un index secondaire est défini dans une table. Un index secondaire global peut utiliser un attribut scalaire comme clé de hachage ou de plage. L'extraction d'éléments à l'aide d'index secondaires est réalisée à l'aide de l'interface de requête définie dans l'API DynamoDB.

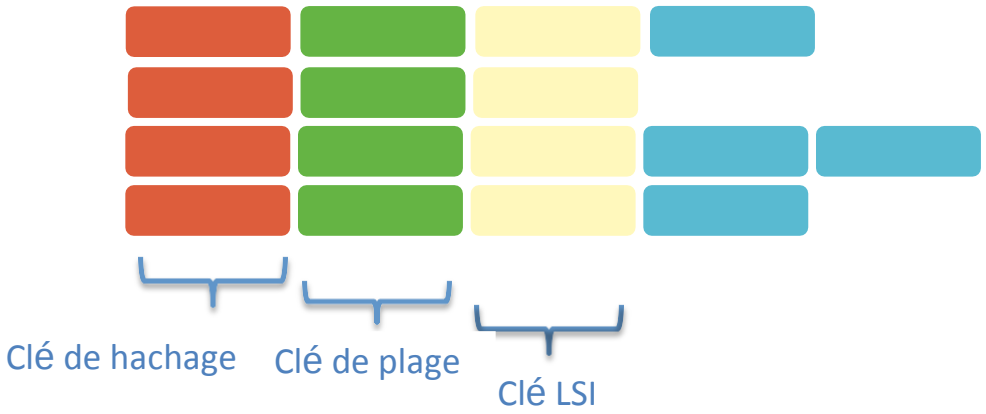

Figure 5 : Index secondaire local

Étant donné que le nombre d'index locaux et globaux pouvant exister par table est limité, il est important de comprendre totalement les exigences en matière d'accès aux données des applications qui utilisent DynamoDB pour le stockage permanent. De plus, des index secondaires globaux requièrent que des valeurs d'attribut soient projetées sur l'index. Cela signifie que lorsqu'un index est créé, un sous-ensemble d'attributs de la table parent doit être sélectionné pour être inclus dans l'index. Lorsqu'un élément est interrogé à l'aide d'un index secondaire global, les seuls attributs qui seront remplis dans l'élément retourné sont ceux qui ont été projetés dans l'index. La figure 6 présente ce concept. Notez comment les attributs originaux de clé de hachage et de plage sont automatiquement promus dans l'index secondaire global. Les lectures sur des index secondaires globaux sont toujours cohérentes à terme, alors que des index secondaires locaux prennent en charge la cohérence à terme ou forte. Enfin, les index secondaires locaux et globaux utilisent des E/S provisionnées (présentées en détails ci-dessous) pour des lectures et des écritures sur l'index. Cela signifie qu'à chaque fois qu'un élément est inséré ou mis à jour dans la table principale, des index secondaires consommeront des E/S pour mettre à jour l'index.

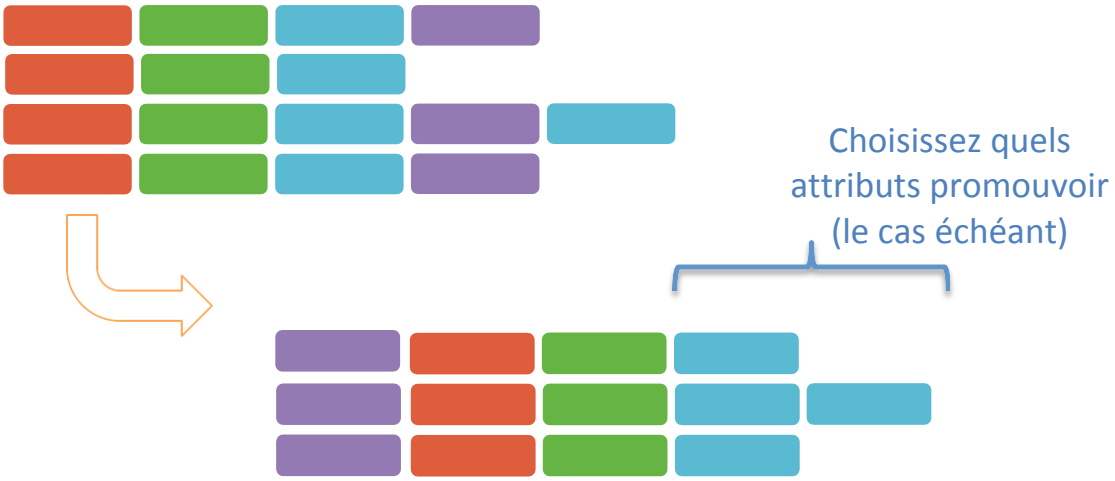

Figure 6 : Création d'un index secondaire global dans une table

Lorsqu'un élément est lu ou écrit sur une table ou un index DynamoDB, le volume de données requis pour effectuer l'opération de lecture ou d'écriture est exprimé comme unité de lecture ou d'écriture. Une unité de lecture comporte 4 K de données et une unité d'écriture comporte 1 K. Cela signifie que l'extraction d'un élément d'une taille de 8 K consommera 2 unités de lecture de données. L'insertion de l'élément consommera 8 unités d'écriture de données. Le nombre d'unités de lecture et d'écriture autorisées par seconde correspond aux E/S provisionnées de la table. Si votre application nécessite que 1 000 éléments de 4 K soient écrits par seconde, la capacité d'écriture provisionnée de la table devrait être d'au moins 4 000 unités d'écriture par seconde. Lorsqu'un volume insuffisant de capacité de lecture ou d'écriture est provisionné dans une table, le service DynamoDB limitera les opérations de lecture et d'écriture. Cela peut entraîner des performances faibles et, dans certains cas, des exceptions de limitation dans l'application cliente. Par conséquent, il est important de comprendre les exigences d'E/S d'une application lors de la conception des tables. Toutefois, la capacité en lecture et en écriture peut être altérée dans une table existante, et si une application connaît un pic soudain de l'utilisation qui entraîne des limites, les E/S provisionnées peuvent être augmentées pour gérer la nouvelle charge de travail. De même, si la charge diminue pour quelque raison que ce soit, les E/S provisionnées peuvent être diminuées. Cette capacité à modifier de manière dynamique les caractéristiques d'E/S d'une table est un facteur de différentiation clé entre DynamoDB et un SGBDR classique, dans lequel le débit des E/S doit être fixé sur base du matériel sous-jacent sur lequel le moteur de base de données est exécuté.

Migration du SGBDR vers DynamoDB

Dans la section précédente, nous avons discuté de certaines des principales fonctions de DynamoDB, ainsi que de certaines différences essentielles entre DynamoDB et un SGBDR classique. Dans cette section, nous proposerons une stratégie pour la migration d'un SGBDR vers DynamoDB qui tient compte de ces fonctions et différences clés. Dans la mesure où les migrations de bases de données sont généralement complexes et risquées, nous vous recommandons d'opter pour une approche répétitive par étape. Comme c'est le cas avec l'adoption de toute nouvelle technologie, il est également intéressant de se concentrer d'abord sur les cas d'utilisation les plus faciles. Il est également important de se rappeler que, comme nous le proposons dans cette section, la migration vers DynamoDB ne doit pas être un processus extrême basé sur le tout ou rien. Pour certaines migrations, il est possible d'exécuter la charge de travail sur DynamoDB et le SGBDR en parallèle, et de passer à DynamoDB uniquement lorsqu'il est clair que la migration a réussi et que l'application fonctionne correctement.

Le diagramme d'état suivant présente notre stratégie de migration proposée :

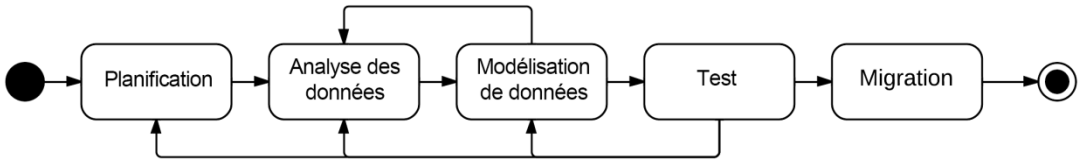

Figure 7 : Phases de migration

Il est important de noter que ce processus est répétitif. Le résultat de certains états peut entraîner un retour à un état précédent. Les problèmes de la phase d'analyse et de modélisation des données peuvent ne pas survenir avant les tests. Dans la plupart des cas, il sera nécessaire de répéter ces phases plusieurs fois avant d'atteindre l'état final de migration des données. Chaque phase sera présentée en détails dans les sections suivantes.

Phase de planification

La première partie de la phase de planification est l'identification des objectifs de la migration des données. Il s'agit notamment (mais sans s'y limiter) des objectifs suivants :

- Augmentation des performances des applications
- Réduction des coûts
- Diminution de la charge sur un SGBDR

Dans de nombreux cas, les objectifs d'une migration peuvent être une combinaison de tous les points ci-dessus. Une fois ces objectifs définis, ils peuvent être utilisés pour informer de l'identification des tables de SGBDR à migrer vers DynamoDB. Comme nous l'avons mentionné précédemment, des tables utilisées par des charges de travail impliquant des données non relationnelles constituent les meilleurs choix pour la migration vers DynamoDB. La migration de ce type de tables vers DynamoDB peut entraîner des performances d'application sensiblement accrues, ainsi qu'une diminution des coûts et des charges sur le SGBDR. Certains bons candidats pour la migration sont les suivants :

- Tables Entité-attribut-valeur
- Tables d'état de session d'application
- Tables des préférences de l'utilisateur
- Tables de journalisation

Une fois les tables identifiées, les caractéristiques des tables sources qui rendent la migration difficile doivent être documentées. Ces informations seront essentielles pour choisir une stratégie de migration solide. Examinons quelques-uns des principaux défis qui doivent avoir un impact sur la stratégie de migration :

Défi	Impact sur la stratégie de migration
Les écritures sur la table source du SGBDR ne peuvent pas être choisies avant ou pendant la migration.	La synchronisation des données dans la table DynamoDB cible avec la source sera difficile. Pensez à une stratégie de migration qui implique l'écriture de données dans les tables sources et cibles en parallèle.
Le volume de données dans la table source dépasse ce qui peut être raisonnablement transféré avec la bande passante réseau existante.	Pensez à exporter les données depuis la table source vers des disques amovibles et à l'aide du service AWS Import/Export pour importer les données vers un compartiment dans S3. Importez ces données dans DynamoDB directement depuis S3. Vous pouvez également réduire le volume de données qui doit être migré en exportant uniquement les enregistrements qui ont été créés après un point temporel récent. Toutes les données antérieures à ce point resteront dans la table existante dans le SGBDR.
Les données dans la table source doivent être transformées avant de pouvoir être importées dans DynamoDB.	Exportez les données de la table source et transférez-les vers S3. Pensez à utiliser EMR pour la transformation des données et importez les données transformées dans DynamoDB.
La structure de clé primaire de la table source ne peut pas être transférée vers DynamoDB.	Identifiez la ou les colonnes qui permettront d'utiliser les clés de hachage et de plage pour les éléments importés. Pensez également à ajouter une clé de substitution (comme un UUID) à la table source qui fera office de clé de hachage appropriée.

Défi	Impact sur la stratégie de migration
Les données dans la table source sont chiffrées.	Si le chiffrement est géré par le SGBDR, les données devront être déchiffrées lors de leur exportation, puis rechiffrées pendant l'importation à l'aide d'un schéma de chiffrement 0.55 cmutilisé par l'application, et pas par le moteur de base de données sous-jacent. Les clés cryptographiques devront être gérées en dehors de DynamoDB.

Tableau 1 : Défis ayant un impact sur la stratégie de migration

Enfin, et peut-être l'élément le plus important, le processus de sauvegarde et de récupération doit être défini et documenté pendant la phase de planification. Si la stratégie de migration nécessite un transfert complet du SGBDR vers DynamoDB, il est essentiel de définir un processus pour la fonctionnalité de restauration à l'aide du SGBDR en cas de problème de migration. Pour atténuer les risques, pensez à exécuter la charge de travail sur DynamoDB et le SGBDR en parallèle pendant une certaine période. Dans ce scénario, l'application du SGBDR existante peut être désactivée seulement une fois que la charge de travail a été suffisamment testée en production à l'aide de DynamoDB.

Phase d'analyse des données

L'objectif de la phase d'analyse des données est de comprendre la composition des données source et d'identifier les modèles d'accès aux données utilisés par l'application. Ces informations sont des entrées requises dans la phase de modélisation des données. C'est également essentiel pour comprendre les coûts et les performances de l'exécution d'une charge de travail dans DynamoDB. L'analyse des données source devrait inclure :

- Une estimation du nombre d'éléments à importer dans DynamoDB
- Une distribution des tailles d'élément
- La multiplicité des valeurs à utiliser comme clés de hachage ou de plage

La tarification de DynamoDB comporte deux composants principaux : le stockage et les E/S provisionnées. Les exigences en matière de stockage et d'E/S provisionnées pour la table peuvent être calculées en estimant le nombre d'éléments qui seront importés dans une table DynamoDB et la taille approximative de chaque élément. Des types de données SQL communs seront mappés sur l'un des trois types scalaires dans DynamoDB : chaîne, nombre et binaire. La longueur du type de données « nombre » est variable et les chaînes sont codées à l'aide du format UTF-8 binaire. Il convient de se concentrer sur les attributs présentant les valeurs les plus importantes lors de l'estimation de la taille de l'élément, car des IOPS provisionnées sont données par incréments intégraux de 1 K ; il n'existe pas de concept d'E/S fractionnelles dans DynamoDB. Si la taille d'un élément est estimée à 3,3 K, il aura besoin de 4 unités d'E/S d'écriture de 1 K et d'une unité d'E/S de lecture de 4 K pour lire et écrire un seul élément, respectivement. Étant donné que la taille sera arrondie au kilo-octet le plus proche, la taille exacte des types numériques n'est pas importante. Dans la majorité des cas, même pour de grands nombres avec une précision élevée, les données seront stockées à l'aide d'un petit nombre d'octets. Étant donné que chaque élément dans une table peut contenir un nombre variable d'attributs, il est utile de calculer une répartition des tailles d'élément et d'utiliser une valeur de percentile pour estimer la taille de l'élément. Par exemple, vous pouvez choisir une taille d'élément représentant le 95_e percentile et utiliser cette information pour estimer les coûts de stockage et d'E/S provisionnées. S'il y a trop de lignes dans la table source pour permettre un contrôle individuel, prenez des exemples de données source et utilisez-les pour calculer la répartition de la taille d'élément.

À tout le moins, une table doit disposer d'unités de lecture et d'écriture suffisamment provisionnées pour lire et écrire un seul élément par seconde. Par exemple, si 4 unités d'écriture sont requises pour écrire un élément d'une taille égale ou inférieure au 95_e percentile, la table doit avoir au moins 4 unités d'écriture d'E/S provisionnées par seconde. Une taille plus petite et l'écriture d'un seul élément entraîneront des limitations et dégraderont les performances. En pratique, le nombre d'unités de lecture et d'écriture provisionnées sera plus grand que le minimum requis. Une application qui utilise DynamoDB pour le stockage de données devra généralement effectuer des lectures et des écritures simultanément.

L'estimation correcte des E/S provisionnées est essentielle pour garantir les performances requises de l'application et pour comprendre les coûts. Il est essentiel de comprendre la fréquence de distribution des valeurs de colonne du SGBDR qui peuvent être des clés de hachage ou de plage pour obtenir également les performances maximales. Les colonnes contenant des valeurs qui ne sont pas réparties uniformément (c.-à-d. certaines valeurs apparaissent en nombre bien plus élevé que d'autres) ne sont pas de bonnes clés de hachage ou de plage car l'accès aux éléments avec des clés présentant une fréquence élevée touchera les mêmes partitions de DynamoDB, et cela présente des implications négatives au niveau des performances.

Le deuxième objectif de la phase d'analyse des données est de catégoriser les modèles d'accès aux données de l'application. Étant donné que DynamoDB ne prend pas en charge un langage de requête générique comme SQL, il est essentiel de documenter les manières dont les données seront écrites et lues depuis les tables. Ces informations sont essentielles pour la phase de modélisation des données pendant laquelle les tables, la structure des clés et les index seront définis. Certains modèles fréquents d'accès aux données sont les suivants :

- Écriture seule : les éléments sont écrits sur une table et ne sont jamais lus par l'application.
- Extractions par des valeurs distinctes : les éléments sont extraits individuellement par une valeur qui identifie de manière unique l'élément dans la table.
- Requêtes sur une plage de valeurs : c'est fréquent avec des données temporelles.

Comme nous le verrons dans la section suivante, la documentation de modèles d'accès aux données d'une application à l'aide de catégories comme celles décrites ci-dessus permettra de prendre la majorité des décisions en matière de modélisation de données.

Phase de modélisation de données

Lors de cette phase, les tables, les clés de hachage et de plage et les index secondaires seront définis. Le modèle de données produit dans le cadre de cette phase doit prendre en charge les modèles d'accès aux données décrits dans la phase d'analyse des données. La première étape de modélisation des données est de déterminer les clés de hachage et de plage pour une table. La clé primaire, composée de la clé de hachage ou d'un composite de clés de hachage et de plage, doit être unique pour tous les éléments de la table. Lors de la migration de données depuis un SGBDR, il est tentant d'utiliser la clé primaire de la table source comme clé de hachage. Mais en réalité, cette clé présente souvent une importance sémantique moindre pour l'application. Par exemple, une table des utilisateurs dans un SGBDR peut définir une clé primaire numérique, mais une application responsable de la journalisation d'un utilisateur demandera une adresse électronique, et pas l'ID d'utilisateur numérique. Dans ce cas, l'adresse électronique est la clé naturelle et serait mieux adaptée comme clé de hachage dans la table DynamoDB, car des éléments peuvent être facilement extraits par leurs valeurs de clé de hachage. La modélisation de la clé de hachage de cette manière convient aux modèles d'accès aux données qui extraient des éléments par valeur distincte. Pour d'autres modèles d'accès aux données, comme en écriture seule, l'utilisation d'un ID numérique généré de manière aléatoire fonctionnera bien pour la clé de hachage. Dans ce cas, les éléments ne seront jamais extraits de la table par l'application, et donc, la clé ne sera utilisée que pour identifier de façon unique les éléments, et pas comme un moyen d'extraire les données.

Les tables du SGBDR qui contiennent un index unique sur deux valeurs clés sont idéales pour définir une clé primaire à l'aide d'une clé de hachage et d'une clé de plage. Les tables d'intersection utilisées pour définir des relations entre de nombreux éléments dans un SGBDR sont généralement modélisées à l'aide d'un index unique sur les valeurs clés des deux côtés de la relation. Étant donné que l'extraction de données dans une relation entre de nombreux éléments nécessite une série de jointures de table, la migration de ce type de table vers DynamoDB impliquerait également la dénormalisation des données (présentée plus en détails ci-dessous). Les valeurs de date sont également souvent utilisées comme clés de plage. Une table comptant le nombre de visites d'une URL un jour donné pourrait définir l'URL comme clé de hachage et la date comme clé de plage. Comme avec des clés primaires composées uniquement d'une clé de hachage, l'extraction d'éléments avec une clé primaire composite nécessite que l'application spécifie les valeurs de clé de hachage et de plage. Cela doit être pris en compte lorsque vous évaluez si une clé de substitution ou une clé naturelle serait mieux adaptée pour la clé de hachage ou de plage.

Étant donné que des attributs non clés peuvent être ajoutés arbitrairement à un élément, les seuls attributs qui doivent être spécifiés dans une définition de table DynamoDB sont la clé de hachage et (voire) la clé de plage. Toutefois, si des index secondaires vont être définis sur des attributs non clés, ils doivent être inclus dans la définition de table. L'inclusion d'attributs non clés dans la définition de table n'impose pas de type de schéma pour tous les éléments de la table. Outre la clé primaire, chaque élément de la table peut disposer d'une liste arbitraire d'attributs.

L'assistance pour SQL dans un SGBDR signifie que des enregistrements peuvent être extraits à l'aide des valeurs de colonne dans la table. Ces requêtes peuvent ne pas toujours être efficaces ; si aucun index n'existe dans la colonne utilisée pour extraire les données, une analyse complète de la table peut être requise pour localiser les lignes correspondantes. L'interface de requête exposée par l'API de DynamoDB ne prend pas en charge l'extraction d'éléments depuis une table de cette manière. Il est possible de réaliser une analyse complète de la table, mais c'est inefficace et cela consomme de nombreuses unités de lecture si la table est volumineuse. Au lieu de cela, des éléments peuvent être extraits d'une table DynamoDB par la clé primaire de la table ou la clé d'un index secondaire local ou global définie dans la table. Étant donné qu'un index dans une colonne non clé d'une table de SGBDR suggère que l'application interroge fréquemment les données sur cette valeur, ces attributs sont idéaux pour des index secondaires locaux ou globaux dans une table DynamoDB. Le nombre d'index secondaires autorisés sur une table DynamoDB[2] est limité. Il est donc important de choisir de définir des clés pour ces index à l'aide de valeurs d'attribut que l'application utilisera le plus fréquemment pour l'extraction de données.

[2] http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Limits.html

DynamoDB ne prend pas en charge le concept de jointure de table. Par conséquent, la migration de données depuis une table de SGDBR nécessitera souvent une dénormalisation des données. Pour les utilisateurs habitués à un SGBDR, il s'agit ici d'un concept étranger, voire inconfortable. Étant donné que les charges de travail le mieux adaptées pour la migration vers DynamoDB depuis un SGBDR ont tendance à impliquer des données non relationnelles, la dénormalisation présente rarement les mêmes problèmes que dans un modèle de données relationnelles. Par exemple, si une base de données relationnelle comporte une table User et UserAddress, associée à l'ID d'utilisateur, vous combinerez les attributs d'utilisateur avec ceux d'adresse dans une seule table DynamoDB. Dans la base de données relationnelle, la normalisation des informations UserAddress permet de spécifier plusieurs adresses pour un utilisateur donné. Il s'agit d'une exigence pour un système de gestion de contacts ou de CRM. Mais dans DynamoDB, une table User devrait avoir un autre objectif, peut-être le suivi des utilisateurs enregistrés d'une application mobile. Dans ce cas d'utilisation, la multiplicité des utilisateurs et des adresses est moins importante que la flexibilité et la récupération rapide d'enregistrements d'utilisateur.

Exemple de modélisation de données

Passons à un exemple qui combine les concepts décrits dans cette section et la précédente. Cet exemple montrera comment utiliser des index secondaires pour un accès efficace aux données, et comment estimer la taille de l'élément et le volume requis d'E/S provisionnées pour une table DynamoDB. La Figure 8 présente un diagramme ERD pour un schéma utilisé pour suivre des événements lors du traitement de commandes réalisées en ligne via un portail d'e-commerce. Les structures de table du SGBDR et de DynamoDB sont illustrées.

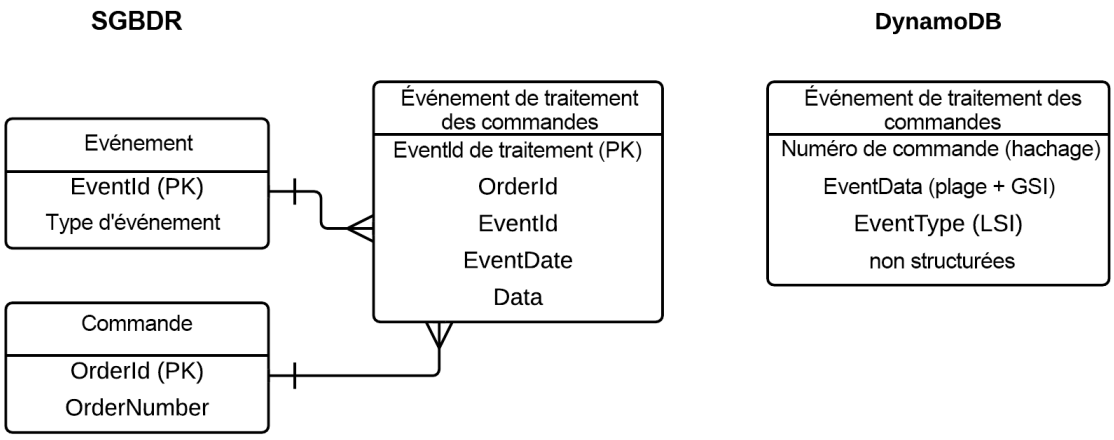

Figure 8 : Schéma du SGBDR et de DynamoDB pour le suivi d'événements

Le nombre de lignes qui seront migrées est d'environ 10^8. Par conséquent, le calcul du 95e percentile de la taille de l'élément n'est pas pratique. À la place, nous réaliserons un échantillonnage aléatoire simple avec le remplacement de 10^6 lignes. Cela nous donnera la précision adéquate pour estimer la taille de l'élément. Pour ce faire, nous créerons une vue SQL qui contient les champs qui seront insérés dans la table DynamoDB. Ensuite, notre méthode d'échantillonnage sélectionne de manière aléatoire 10^6 lignes dans cette vue et calcule la taille qui représente le 95e percentile.

Cet échantillonnage statistique utilise une taille de 95e percentile de 6,6 Ko, dont la majorité est utilisée par l'attribut Data (qui peut faire une taille de 6 Ko). Le nombre minimal d'unités d'écriture requis pour écrire un seul élément est :

$$\text{plafond (6,6 Ko par élément/1 Ko par unité d'écriture)} = \text{7 unités d'écriture par élément}$$

Le nombre minimal d'unités de lecture requises pour lire un seul élément est calculé de la même manière :

$$\text{plafond (6,6 Ko par élément/4 Kb par unité de lecture)} = \text{2 unités de lecture par élément}$$

Cette charge de travail spécifique présente une forte densité d'écritures et nous avons besoin d'un nombre suffisant d'E/S pour écrire 1 000 événements pour 500 commandes par jour. Cette charge est calculée comme suit :

$$\text{500 commandes par jour} * \text{1 000 événements par commande} = 5 * 10^5 \text{ événements par jour}$$

$$5 * 10^5 \text{ événements par jour} * \text{86 400 secondes par jour} = \text{5,78 événements par seconde}$$

$$\text{plafond (5,78 événements par seconde} * \text{7 unités d'écriture par élément)} = \text{41 unités d'écriture par seconde}$$

Les lectures dans la table ne surviennent qu'une seule fois par heure, lorsque les données de l'heure précédente sont importées dans un cluster Amazon Elastic Map Reduce pour ETL. Cette opération utilise une requête qui sélectionne des éléments depuis une plage de données spécifique (c'est pourquoi l'attribut EventDate est une clé de plage et un index secondaire global). Le nombre d'unités de lecture (qui sera provisionné sur l'index secondaire global) requises pour récupérer les résultats d'une requête est basé sur la taille des résultats retournés par la requête :

$$\text{5,78 événements par seconde} * \text{3 600 secondes par heure} = \text{20 808 événements par heure}$$

$$\frac{\text{20 808 événements par heure} * \text{6,6 Ko par élément}}{\text{1 024 Ko}} = \text{134,11 Mo par heure}$$

Le volume maximal de données retourné dans une seule opération de requête est de 1 Mo. Par conséquent, la pagination sera requise. Chaque requête de lecture horaire nécessitera la lecture de 135 pages de données. Pour des lectures à cohérence forte, 256 unités de lecture sont requises pour lire une page complète simultanément (le nombre correspond à la moitié des lectures à cohérence à terme). Par conséquent, pour prendre en charge cette charge de travail spécifique, 256 unités de lecture et 41 unités d'écriture seront requises. D'un point de vue pratique, les unités d'écriture seraient probablement exprimées par un nombre pair, comme 48. Nous disposons désormais de toutes les données dont nous avons besoin pour estimer le coût de DynamoDB pour cette charge de travail :

1. Nombre d'éléments (10^8)
2. Taille de l'élément (7 Ko)
3. Unités d'écriture (48)
4. Unités de lecture (256)

Ils peuvent être exécutés via le Calculateur de coûts mensuels Amazon[3] pour tirer profit d'une estimation de coût.

Phase de test

La phase de test est la partie la plus importante de la stratégie de migration. Pendant cette phase, l'ensemble du processus de migration sera testé de bout en bout. Un plan de test exhaustif devrait au moins comporter les éléments suivants :

[3] http://calculator.s3.amazonaws.com/index.html

Catégorie test	Objectif
Tests d'acceptation de base	Ces tests doivent être exécutés automatiquement lors de la réalisation des routines de migration des données. Leur objectif principal est de vérifier si la migration des données a fonctionné correctement. Voici quelques résultats courants de ces tests : • Nombre total d'éléments traités • Nombre total d'éléments importés • Nombre total d'éléments ignorés • Nombre total d'avertissements • Nombre total d'erreurs Si l'un de ces totaux issus des tests ne correspond pas aux valeurs attendues, cela signifie que la migration a échoué et que des problèmes doivent être résolus avant le passage à l'étape suivante du processus ou au prochain cycle de test.
Tests fonctionnels	Ces exercices testent les fonctionnalités de la ou des applications à l'aide de DynamoDB pour le stockage des données. Ils comprendront une combinaison de tests manuels et automatisés. L'objectif principal des tests fonctionnels est d'identifier les problèmes de l'application entraînés par la migration des données du SGBDR dans DynamoDB. C'est pendant cette série de tests que des failles du modèle de données sont souvent découvertes.
Tests non fonctionnels	Ces tests évaluent les caractéristiques non fonctionnelles de l'application, par exemple les performances avec différents niveaux de chargement et la résistance aux failles d'une partie de la pile d'application. Ces tests peuvent également inclure des cas de limite ou périphériques peu probables mais qui pourraient avoir un impact négatif sur l'application (par exemple, si un grand nombre de clients essaient de mettre à jour le même enregistrement exactement au même moment). Le processus de sauvegarde et de récupération défini pendant la phase de planification doit également être inclus dans un test non fonctionnel.
Tests d'acceptation utilisateur	Ces tests doivent être exécutés par les utilisateurs finaux de la ou des applications une fois la migration définitive des données terminée. L'objectif de ces tests est de permettre aux utilisateurs finaux de décider si l'application peut être utilisée pour sa fonctionnalité principale au sein de l'organisation.

Tableau 2 : Plan de test de migration des données

Étant donné que la stratégie de migration est répétitive, ces tests seront réalisés à plusieurs reprises. Pour une efficacité maximale, pensez à tester les routines de migration de données à l'aide d'un échantillonnage de données de production si le volume total de données à migrer est important. Le résultat de la phase de test nécessitera souvent de revoir une phase précédente pendant le processus. La stratégie de migration générale sera plus affinée lors de chaque itération pendant le processus. Une fois tous les tests exécutés correctement, cela indique qu'il est temps de passer à la phase suivante, la phase finale : la migration des données.

Phase de migration de données

Dans la phase de migration des données, l'ensemble des données de production provenant des tables du SGBDR source seront migrées dans DynamoDB. Lorsque cette phase est atteinte, le processus de migration des données de bout en bout aura été testé et analysé de manière détaillée. Toutes les étapes du processus auront été correctement documentées. Par conséquent, l'exécution du processus sur l'ensemble de données de production doit être aussi simple que l'application d'une procédure qui a été exécutée à de nombreuses reprises auparavant. En préparation de cette phase finale, une notification doit être envoyée aux utilisateurs de l'application, les avertissant que l'application réalisera une maintenance et devrait connaître (si nécessaire) un temps d'arrêt.

Une fois la migration des données réalisée, les tests d'acceptation utilisateur définis dans la phase précédente doivent être exécutés une dernière fois pour veiller à ce que l'application soit utilisable. Si la migration échoue pour quelque raison que ce soit, la procédure de sauvegarde et de récupération, qui aura également été testée et analysée de manière détaillée d'ici là, peut être exécutée. Lorsque le système revient à un état stable, une analyse de la cause racine de la défaillance doit être réalisée et la migration de données doit être replanifiée une fois la cause racine résolue. Si tout se passe bien, l'application doit être surveillée de manière étroite pendant les jours qui suivent jusqu'à ce qu'il y ait suffisamment de données indiquant que l'application fonctionne normalement.

Conclusion

L'utilisation de DynamoDB pour des charges de travail appropriées peut entraîner une diminution des coûts, une réduction de la surcharge opérationnelle et une augmentation des performances, de la disponibilité et de la fiabilité par rapport à un SGBDR classique. Dans ce document, nous avons proposé une stratégie pour l'identification et la migration de charges de travail appropriées d'un SGBDR vers DynamoDB. Alors que la mise en œuvre d'une telle stratégie nécessitera un effort de planification et d'ingénierie prudent, nous sommes confiants dans le fait que le retour sur investissement de la migration vers une solution NoSQL totalement opérée comme DynamoDB dépassera de loin le coût initial associé à cet effort.

Fiche de révision

Voici une fiche de révision qui résume certains concepts clés présentés dans ce document et les sections dans lesquelles ces concepts sont détaillés :

Concept	Section
Détermination de charges de travail adaptées	*Charges de travail adaptées*
Choix de la structure de clé appropriée	*Concepts clés*
Indexation de table	*Phase de modélisation de données*
Allocation du débit d'écriture et de lecture	*Exemple de modélisation de données*
Sélection d'une stratégie de migration	*Phase de planification*

Suggestions de lecture

Pour obtenir de l'aide, consultez les ressources suivantes :

- Guide du développeur DynamoDB[4]
- Site Web DynamoDB[5]

[4] http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStartedDynamoDB.html
[5] http://aws.amazon.com/dynamodb