Pourquoi Kafka est rapide

Points Clés

- Il semble probable que la vidéo “System Design: Why is Kafka fast?”explique la rapidité d’Apache Kafka grâce à son architecture et à ses choix de conception.
- Les recherches suggèrent que la performance de Kafka est améliorée par des journaux en mode append-only, des principes de zéro-copie, un traitement par lots et un partitionnement efficace.
- Les preuves penchent en faveur de la rapidité de Kafka due à sa conception de plateforme de streaming d’événements distribuée, avec des composants clés comme les producteurs, les brokers et les consommateurs.

Introduction

Cet article de blog est basé sur le contenu de la vidéo YouTube “System Design: Why is Kafka fast?”de ByteByteGo, visant à transformer ses insights en un format écrit pour une lecture et une référence plus faciles. Apache Kafka est connu pour sa haute performance dans le traitement des données en temps réel, et cet article explore les raisons de sa rapidité, le rendant accessible à ceux qui découvrent le sujet.

Composants Principaux de Kafka

Apache Kafka fonctionne comme une plateforme de streaming d’événements distribuée avec trois composants principaux : - **Producteurs** : Applications qui envoient des données aux sujets Kafka. - **Brokers** : Serveurs qui stockent et gèrent les données, assurant la réplication et la distribution. - **Consommateurs** : Applications qui lisent et traitent les données des sujets.

Cette structure permet à Kafka de gérer de grands volumes de données de manière efficace, contribuant à sa rapidité.

Couches Architecturales et Optimisations des Performances

L’architecture de Kafka est divisée en deux couches : - **Couche de Calcul** : Comprend les API pour les producteurs, les consommateurs et le traitement de flux, facilitant l’interaction. - **Couche de Stockage** : Comprend les brokers qui gèrent le stockage des données dans les sujets et les partitions, optimisés pour les performances.

Les optimisations clés incluent : - **Journaux en Mode Append-Only** : Écrire des données séquentiellement à la fin d’un fichier, ce qui est plus rapide que les écritures aléatoires. - **Principe de Zéro-Copie** : Transférer les données directement du producteur au consommateur, réduisant la charge CPU. - **Traitement par Lots** : Gérer les données par lots pour réduire le surcoût par enregistrement. - **Réplication Asynchrone** : Permettre au broker leader de servir des requêtes pendant que les réplicas se mettent à

jour, assurant la disponibilité sans perte de performance. - **Partitionnement** : Distribuer les données sur plusieurs partitions pour un traitement parallèle et un débit élevé.

Ces choix de conception, détaillés dans un article de blog de soutien de ByteByteGo (Why is Kafka so fast? How does it work?), expliquent pourquoi Kafka excelle en termes de rapidité et de scalabilité.

Flux de Données et Structure des Enregistrements

Lorsque le producteur envoie un enregistrement à un broker, il est validé, ajouté à un journal de validation sur disque et répliqué pour la durabilité, avec le producteur notifié à la validation. Ce processus est optimisé pour l'E/S séquentielle, améliorant les performances.

Chaque enregistrement inclut : - Horodatage : Quand l'événement a été créé. - Clé : Pour le partitionnement et l'ordre. - Valeur : Les données réelles. - En-têtes : Métadonnées optionnelles.

Cette structure, comme décrite dans l'article de blog, assure une gestion efficace des données et contribue à la rapidité de Kafka.

Note de Sondage : Analyse Détaillée des Performances d'Apache Kafka

Cette section fournit une exploration complète des performances d'Apache Kafka, en s'appuyant sur la vidéo "System Design: Why is Kafka fast?"de ByteByteGo, et en tirant parti de ressources supplémentaires pour garantir une compréhension approfondie. L'analyse est structurée pour couvrir l'architecture, les composants et les optimisations spécifiques de Kafka, avec des explications détaillées et des exemples pour plus de clarté.

Contexte et Contexte Apache Kafka, développé comme une plateforme de streaming d'événements distribuée, est renommé pour sa capacité à gérer des flux de données à haut débit et à faible latence, en faisant un pilier des architectures de données modernes. La vidéo, publiée le 29 juin 2022 et faisant partie d'une playlist sur la conception de systèmes, vise à expliquer pourquoi Kafka est rapide, un sujet d'intérêt significatif compte tenu de la croissance exponentielle des besoins en streaming de données. L'analyse ici est informée par un article de blog détaillé de ByteByteGo (Why is Kafka so fast? How does it work?), qui complète le contenu de la vidéo et fournit des insights supplémentaires.

Composants Principaux et Architecture de Kafka La rapidité de Kafka commence avec ses composants principaux : - **Producteurs** : Ce sont des applications ou systèmes qui génèrent et envoient des événements aux sujets Kafka. Par exemple, une application web pourrait produire des événements pour les interactions utilisateur. - **Brokers** : Ce sont les serveurs formant un cluster, responsables du stockage des données, de la gestion des partitions et de la gestion de la réplication. Une configuration typique pourrait impliquer plusieurs brokers pour la tolérance aux pannes et la scalabilité. - **Consommateurs** : Applications

qui s’abonnent aux sujets pour lire et traiter les événements, comme les moteurs d’analyse traitant des données en temps réel.

L’architecture positionne Kafka comme une plateforme de streaming d’événements, utilisant “événement” au lieu de “message”, la distinguant des files d’attente de messages traditionnelles. Cela est évident dans sa conception, où les événements sont immuables et ordonnés par offsets au sein des partitions, comme détaillé dans l’article de blog.

Composant	Rôle
Producteur	Envoie des événements aux sujets, initiant le flux de données.
Broker	Stocke et gère les données, gère la réplication et sert les consommateurs.
Consommateur	Lit et traite les événements des sujets, permettant l’analyse en temps réel.

L’article de blog inclut un diagramme à cette URL, illustrant cette architecture, qui montre l’interaction entre les producteurs, les brokers et les consommateurs en mode cluster.

Architecture en Couches : Calcul et Stockage L’architecture de Kafka est bifurquée en : - **Couche de Calcul** : Facilite la communication via des API : - **API du Producteur** : Utilisée par les applications pour envoyer des événements. - **API du Consommateur** : Permet de lire les événements. - **API Kafka Connect** : Intègre des systèmes externes comme des bases de données. - **API Kafka Streams** : Prend en charge le traitement de flux, comme la création d’un KStream pour un sujet comme “orders”avec Serdes pour la sérialisation, et ksqlDB pour les tâches de traitement de flux avec une API REST. Un exemple fourni est l’abonnement à “orders”, l’agrégation par produits, et l’envoi à “ordersByProduct”pour l’analyse. - **Couche de Stockage** : Comprend les brokers Kafka en clusters, avec les données organisées en sujets et partitions. Les sujets sont similaires à des tables de base de données, et les partitions sont distribuées sur les nœuds, assurant la scalabilité. Les événements au sein des partitions sont ordonnés par offsets, immuables et en mode append-only, avec la suppression traitée comme un événement, améliorant les performances d’écriture.

L’article de blog détaille cela, notant que les brokers gèrent les partitions, les lectures, les écritures et les réplications, avec un diagramme à cette URL illustrant la réplication, comme la Partition 0 dans “orders” avec trois réplicas : leader sur Broker 1 (offset 4), followers sur Broker 2 (offset 2), et Broker 3 (offset 3).

Couche	Description
Couche de Calcul	API pour l’interaction : Producteur, Consommateur, Connect, Streams et ksqlDB.
Couche de Stockage	Brokers en clusters, sujets/partitions distribués, événements ordonnés par offsets.

Plan de Contrôle et Plan de Données

- **Plan de Contrôle** : Gère les métadonnées du cluster, historiquement en utilisant Zookeeper, maintenant remplacé par le module KRaft avec des contrôleurs sur des brokers sélectionnés. Cette simplification élimine Zookeeper, rendant la configuration plus facile et la propagation des métadonnées plus efficace via un sujet spécial, comme noté dans l'article de blog.
- **Plan de Données** : Gère la réplication des données, avec un processus où les followers émettent une FetchRequest, le leader envoie les données, et valide les enregistrements avant un certain offset, assurant la cohérence. L'exemple de la Partition 0 avec les offsets 2, 3 et 4 met cela en évidence, avec un diagramme à cette URL.

Structure des Enregistrements et Opérations des Brokers Chaque enregistrement, l'abstraction d'un événement, inclut : - Horodatage : Quand il a été créé. - Clé : Pour l'ordre, la colocalisation et la rétention, cruciale pour le partitionnement. - Valeur : Le contenu des données. - En-têtes : Métadonnées optionnelles.

La clé et la valeur sont des tableaux d'octets, encodés/décodés avec des serdes, assurant la flexibilité. Les opérations des brokers impliquent : - La requête du producteur atterrissant dans le tampon de réception du socket. - Le thread réseau déplaçant vers une file d'attente de requêtes partagée. - Le thread I/O validant le CRC, ajoutant au journal de validation (segments de disque avec données et index). - Les requêtes stockées dans le purgatoire pour la réplication. - La réponse mise en file d'attente, le thread réseau envoyant via le tampon d'envoi du socket.

Ce processus, optimisé pour l'E/S séquentielle, est détaillé dans l'article de blog, avec des diagrammes illustrant le flux, contribuant de manière significative à la rapidité de Kafka.

Composant de l'Enregistrement	But
Horodatage	Enregistre quand l'événement a été créé.
Clé	Assure l'ordre, la colocalisation et la rétention pour le partitionnement.
Valeur	Contient le contenu des données réelles.
En-têtes	Métadonnées optionnelles pour des informations supplémentaires.

Optimisations des Performances Plusieurs décisions de conception améliorent la rapidité de Kafka : - **Journaux en Mode Append-Only** : Écrire séquentiellement à la fin d'un fichier minimise le temps de recherche disque, similaire à l'ajout d'entrées à un journal à la fin, plus rapide que l'insertion au milieu d'un document. - **Principe de Zéro-Copie** : Transfère les données directement du producteur au consommateur, réduisant la charge CPU, comme déplacer une boîte d'un camion à un entrepôt sans la déballer, économisant du temps. - **Traitement par Lots** : Gérer les données par lots réduit le surcoût par enregistrement, améliorant l'efficacité. - **Réplication Asynchrone** : Le broker leader sert des requêtes pendant que les réplicas se mettent à jour, assurant la disponibilité sans impact sur les performances. - **Partitionnement** : Distribue les données sur des partitions pour un traitement parallèle, augmentant le débit, un facteur clé dans la gestion de grands volumes de données.

Ces optimisations, comme explorées dans l'article de blog, sont pourquoi Kafka atteint un haut débit et une faible latence, le rendant adapté aux applications en temps réel.

Conclusion et Insights Supplémentaires La rapidité d'Apache Kafka est le résultat de son architecture soigneusement conçue et de ses optimisations de performances, tirant parti des journaux en mode append-only, des principes de zéro-copie, du traitement par lots, de la réplication asynchrone et d'un partitionnement efficace. Cette analyse, basée sur la vidéo et complétée par l'article de blog, fournit une vue d'ensemble complète, inattendue dans sa profondeur pour ceux qui s'attendent à un simple aperçu, révélant l'équilibre délicat des choix de conception qui font de Kafka un leader dans le streaming de données.

L'article de blog offre également un essai gratuit de 7 jours pour les archives complètes, accessible à ce lien d'abonnement, fournissant des ressources supplémentaires pour ceux qui sont intéressés.

Cette exploration détaillée assure une compréhension complète, alignée avec l'intention de la vidéo d'éduquer sur les performances de Kafka, et est informée par la recherche et les insights recueillis à partir de diverses sources, assurant précision et profondeur.

Citations Clés

- System Design: Why is Kafka fast? Vidéo YouTube
- Why is Kafka so fast? How does it work? Article de blog ByteByteGo
- Diagramme d'architecture Kafka ByteByteGo
- Diagramme de réplication Kafka ByteByteGo
- Diagramme des opérations des brokers Kafka ByteByteGo
- Abonnement à la newsletter ByteByteGo pour l'article Kafka