Calcul Distribué

Qu'est-ce que le Calcul Distribué ?

Le calcul distribué est la méthode permettant à plusieurs ordinateurs de travailler ensemble pour résoudre un problème commun. Il fait apparaître un réseau informatique comme un puissant ordinateur unique qui fournit des ressources à grande échelle pour traiter des défis complexes.

Par exemple, le calcul distribué peut chiffrer de grands volumes de données, résoudre des équations de physique et de chimie avec de nombreuses variables, et rendre des animations vidéo tridimensionnelles de haute qualité. Les systèmes distribués, la programmation distribuée et les algorithmes distribués sont d'autres termes qui se réfèrent tous au calcul distribué.

Avantages du Calcul Distribué

Scalabilité

Les systèmes distribués peuvent croître avec votre charge de travail et vos besoins. Vous pouvez ajouter de nouveaux nœuds, c'est-à-dire plus de dispositifs de calcul, au réseau de calcul distribué lorsque cela est nécessaire.

Disponibilité

Votre système de calcul distribué ne tombera pas en panne si l'un des ordinateurs tombe en panne. La conception montre une tolérance aux pannes car elle peut continuer à fonctionner même si des ordinateurs individuels échouent.

Consistance

Les ordinateurs d'un système distribué partagent des informations et dupliquent les données entre eux, mais le système gère automatiquement la consistance des données entre tous les différents ordinateurs. Ainsi, vous bénéficiez de la tolérance aux pannes sans compromettre la consistance des données.

Transparence

Les systèmes de calcul distribué fournissent une séparation logique entre l'utilisateur et les dispositifs physiques. Vous pouvez interagir avec le système comme s'il s'agissait d'un seul ordinateur sans vous soucier de la configuration et de la mise en place des machines individuelles. Vous pouvez avoir différents matériels, intergiciels, logiciels et systèmes d'exploitation qui travaillent ensemble pour faire fonctionner votre système de manière fluide.

Efficacité

Les systèmes distribués offrent des performances plus rapides avec une utilisation optimale des ressources matérielles sous-jacentes. En conséquence, vous pouvez gérer n'importe quelle charge de travail sans vous soucier d'une panne système due à des pics de volume ou à une sous-utilisation de matériel coûteux.

Cas d'Utilisation du Calcul Distribué

Le calcul distribué est omniprésent aujourd'hui. Les applications mobiles et web sont des exemples de calcul distribué car plusieurs machines travaillent ensemble en arrière-plan pour que l'application vous donne les informations correctes. Cependant, lorsque les systèmes distribués sont agrandis, ils peuvent résoudre des défis plus complexes. Explorons quelques façons dont différentes industries utilisent des applications distribuées performantes.

Services Financiers

- **Offrir des primes personnalisées à faible coût** en utilisant des systèmes distribués.
- **Utiliser des bases de données distribuées** pour soutenir en toute sécurité un très grand volume de transactions financières.
- **Authentifier les utilisateurs et protéger les clients contre la fraude** avec des systèmes distribués.

Énergie et Environnement

- **Diffuser et consolider les données sismiques** pour la conception structurelle des centrales électriques.
- **Surveillance en temps réel des puits de pétrole** pour une gestion proactive des risques.

Types d'Architecture de Calcul Distribué

En calcul distribué, vous concevez des applications qui peuvent fonctionner sur plusieurs ordinateurs au lieu de sur un seul ordinateur. Vous y parvenez en concevant le logiciel de sorte que différents ordinateurs effectuent différentes fonctions et communiquent pour développer la solution finale. Il existe quatre principaux types d'architecture distribuée.

Architecture Client-Serveur

- **Clients** ont des informations et des capacités de traitement limitées. Ils font des demandes aux serveurs, qui gèrent la plupart des données et autres ressources.
- **Serveurs** synchronisent et gèrent l'accès aux ressources. Ils répondent aux demandes des clients avec des données ou des informations de statut.

Avantages et Limitations

- **Avantages** : Sécurité et facilité de gestion continue.
- **Limitations** : Les serveurs peuvent causer des goulots d'étranglement de communication, surtout lorsque plusieurs machines font des demandes simultanément.

Architecture à Trois Niveaux

- **Machines Clients** : Le premier niveau auquel vous accédez.
- **Serveurs d'Applications** : Le niveau intermédiaire contenant la logique de l'application.
- **Serveurs de Bases de Données** : Le troisième niveau responsable de la récupération et de la consistance des données.

Architecture à N Niveaux

- **Modèles à N Niveaux** : Incluent plusieurs systèmes client-serveur différents communiquant entre eux pour résoudre le même problème.

Architecture Pair-à-Pair

- **Systèmes Pair-à-Pair** : Attribuent des responsabilités égales à tous les ordinateurs du réseau. N'importe quel ordinateur peut remplir toutes les responsabilités.

Comment Fonctionne le Calcul Distribué ?

Le calcul distribué fonctionne en faisant passer des messages entre ordinateurs dans l'architecture des systèmes distribués. Les protocoles de communication ou les règles créent une dépendance entre les composants du système distribué. Cette interdépendance est appelée couplage, et il existe deux principaux types de couplage.

Couplage Faible

Les composants sont faiblement connectés de sorte que les changements apportés à un composant n'affectent pas l'autre. Par exemple, les ordinateurs client et serveur peuvent être faiblement couplés par le temps. Les messages du client sont ajoutés à une file d'attente du serveur, et le client peut continuer à effectuer d'autres fonctions jusqu'à ce que le serveur réponde à son message.

Couplage Étroit

Les systèmes distribués performants utilisent souvent un couplage étroit. Les réseaux locaux rapides connectent généralement plusieurs ordinateurs, créant un cluster. Dans le calcul en cluster, chaque ordinateur est configuré pour effectuer la même tâche. Les systèmes de contrôle central, appelés intergiciels de clustering, contrôlent et planifient les tâches et coordonnent la communication entre les différents ordinateurs.

Calcul Parallèle

Le calcul parallèle est un type de calcul dans lequel un ordinateur ou plusieurs ordinateurs d'un réseau effectuent de nombreux calculs ou processus simultanément. Bien que les termes calcul parallèle et calcul distribué soient souvent utilisés de manière interchangeable, ils présentent certaines différences.

Calcul Parallèle vs. Calcul Distribué

- **Calcul Parallèle** : Tous les processeurs ont accès à une mémoire partagée pour échanger des informations entre eux.
- **Calcul Distribué** : Chaque processeur a une mémoire privée (mémoire distribuée) et utilise le passage de messages pour échanger des informations.

Calcul en Grille

Dans le calcul en grille, des réseaux informatiques géographiquement distribués travaillent ensemble pour effectuer des tâches communes. Une caractéristique des grilles distribuées est que vous pouvez les former à partir de ressources informatiques appartenant à plusieurs individus ou organisations.

Calcul en Grille vs. Calcul Distribué

- **Calcul en Grille** : Calcul distribué à grande échelle qui met l'accent sur la performance et la coordination entre plusieurs réseaux. En interne, chaque grille agit comme un système de calcul étroitement couplé. En externe, les grilles sont plus faiblement couplées. Chaque réseau de grille effectue des fonctions individuelles et communique les résultats aux autres grilles.

Apache Kakfka

Qu'est-ce que le Streaming d'Événements?

Le streaming d'événements (ou "event streaming") est l'équivalent numérique du système nerveux central du corps humain. C'est la base technologique du monde "toujours en marche" où les entreprises sont de plus en plus définies par le logiciel et automatisées, et où l'utilisateur du logiciel est souvent lui-même un autre logiciel.

Techniquement, le streaming d'événements consiste à capturer des données en temps réel provenant de sources telles que des bases de données, des capteurs, des appareils mobiles, des services cloud et des applications logicielles sous forme de flux d'événements. Ces flux sont ensuite stockés de manière durable pour une récupération ultérieure. Les flux d'événements peuvent être manipulés, traités et utilisés en temps réel ou de manière rétrospective, et acheminés vers différentes technologies de destination selon les besoins. Cela garantit un flux continu et une interprétation des données, permettant de disposer de la bonne information au bon endroit et au bon moment.

Utilisations du Streaming d'Événements

Le streaming d'événements est appliqué à une large variété de cas d'utilisation dans de nombreux secteurs et organisations. Voici quelques exemples :

- **Transactions financières en temps réel** : Traitement des paiements et des transactions financières en temps réel, comme dans les bourses, les banques et les assurances.
- **Suivi et surveillance en temps réel** : Suivi et surveillance des voitures, camions, flottes et expéditions en temps réel, comme dans la logistique et l'industrie automobile.

- **Analyse de données de capteurs IoT** : Capture et analyse continues des données de capteurs provenant d'appareils IoT ou d'autres équipements, comme dans les usines et les parcs éoliens.
- **Interactions clients et commandes** : Collecte et réaction immédiate aux interactions clients et aux commandes, comme dans le commerce de détail, l'industrie hôtelière, le voyage et les applications mobiles.
- **Surveillance des patients** : Surveillance des patients dans les hôpitaux et prédiction des changements de condition pour garantir un traitement rapide en cas d'urgence.
- **Connexion des données d'entreprise** : Connexion, stockage et mise à disposition des données produites par différentes divisions d'une entreprise.
- **Plateformes de données et architectures événementielles** : Servir de fondation pour les plateformes de données, les architectures pilotées par les événements et les microservices.

Qu'est-ce qu'Apache Kafka ?

Apache Kafka est une plateforme de streaming d'événements. Cela signifie qu'elle permet de publier, s'abonner, stocker et traiter des flux d'événements de manière continue et fiable. Kafka combine trois capacités clés pour implémenter des cas d'utilisation de streaming d'événements de bout en bout avec une solution éprouvée :

1. **Publication et Abonnement** : Kafka permet de publier (écrire) et de s'abonner (lire) à des flux d'événements, incluant l'importation et l'exportation continue de données depuis et vers d'autres systèmes.
2. **Stockage Durable** : Kafka stocke les flux d'événements de manière durable et fiable aussi longtemps que vous le souhaitez.
3. **Traitement des Événements** : Kafka permet de traiter les flux d'événements en temps réel ou de manière rétrospective.

Kafka offre ces fonctionnalités dans un environnement distribué, hautement évolutif, élastique, tolérant aux pannes et sécurisé. Il peut être déployé sur du matériel physique, des machines virtuelles, des conteneurs, sur site ou dans le cloud. Vous pouvez choisir entre gérer vous-même vos environnements Kafka ou utiliser des services entièrement gérés proposés par divers fournisseurs.

Comment fonctionne Kafka ?

En bref, Kafka est un système distribué composé de serveurs et de clients qui communiquent via un protocole réseau TCP haute performance. Voici un aperçu de ses composants principaux :

Serveurs

- **Cluster Kafka** : Kafka est exécuté sous forme de cluster composé de plusieurs serveurs pouvant s'étendre sur plusieurs centres de données ou régions cloud.
- **Brokers** : Certains de ces serveurs forment la couche de stockage, appelés brokers, qui gèrent le stockage des flux d'événements.
- **Kafka Connect** : D'autres serveurs exécutent Kafka Connect pour importer et exporter des données en continu sous forme de flux d'événements, intégrant Kafka avec vos systèmes existants tels que des bases de données relationnelles ou d'autres clusters Kafka.
- **Tolérance aux Pannes** : Un cluster Kafka est hautement évolutif et tolérant aux pannes. En cas de défaillance d'un serveur, les autres serveurs prendront en charge son travail pour assurer une continuité des opérations sans

perte de données.

Clients

- **Applications et Microservices Distribués** : Les clients permettent d'écrire des applications distribuées et des microservices qui lisent, écrivent et traitent des flux d'événements en parallèle, à grande échelle, et de manière tolérante aux pannes, même en cas de problèmes réseau ou de défaillance de machines.
- **Clients Inclus** : Kafka inclut des clients pour Java et Scala, y compris la bibliothèque Kafka Streams de haut niveau. Il existe également des clients fournis par la communauté Kafka pour Go, Python, C/C++, et bien d'autres langages de programmation, ainsi que des API REST.

Concepts et Terminologie Principaux

Un événement enregistre le fait que "quelque chose s'est passé" dans le monde ou dans votre entreprise. Il est également appelé enregistrement ou message dans la documentation. Lorsque vous lisez ou écrivez des données sur Kafka, vous le faites sous forme d'événements. Conceptuellement, un événement comporte une clé, une valeur, un horodatage et des en-têtes de métadonnées optionnels. Voici un exemple d'événement :

- Clé de l'événement : "Alice"
- Valeur de l'événement : "A effectué un paiement de 200 $ à Bob"
- Horodatage de l'événement : "25 juin 2020 à 14h06"

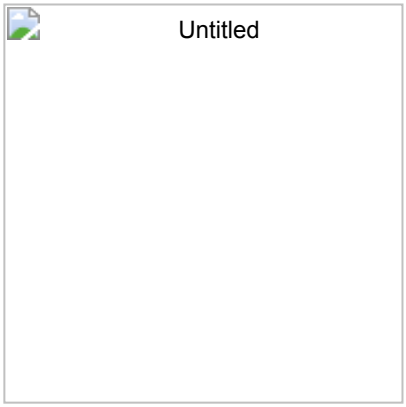
Les producteurs sont des applications clientes qui publient (écrivent) des événements vers Kafka, tandis que les consommateurs sont ceux qui s'abonnent à (lisent et traitent) ces événements. Dans Kafka, les producteurs et les consommateurs sont totalement découplés et agnostiques les uns des autres, ce qui est un élément clé de conception pour atteindre la haute scalabilité pour laquelle Kafka est connu. Par exemple, les producteurs n'ont jamais besoin d'attendre les consommateurs. Kafka offre diverses garanties telles que la capacité de traiter les événements exactement une fois.

Les événements sont organisés et stockés de manière durable dans des topics. Simplifié, un topic est similaire à un dossier dans un système de fichiers, et les événements sont les fichiers dans ce dossier. Un exemple de nom de topic pourrait être "payments". Les topics dans Kafka sont toujours multi-producteurs et multi-consommateurs : un topic peut avoir zéro, un ou plusieurs producteurs qui y écrivent des événements, ainsi que zéro, un ou plusieurs consommateurs qui s'abonnent à ces événements. Les événements dans un topic peuvent être lus aussi souvent que nécessaire : contrairement aux systèmes de messagerie traditionnels, les événements ne sont pas supprimés après consommation. Au lieu de cela, vous définissez la durée pendant laquelle Kafka doit conserver vos événements via un paramètre de configuration par topic, après quoi les anciens événements seront supprimés. Les performances de Kafka sont effectivement constantes par rapport à la taille des données, donc stocker des données pendant une longue période est parfaitement acceptable.

Les topics sont partitionnés, ce qui signifie qu'un topic est réparti sur un certain nombre de "paniers" situés sur différents brokers Kafka. Cette répartition distribuée de vos données est très importante pour la scalabilité car elle permet aux applications clientes de lire et d'écrire les données depuis/vers de nombreux brokers en même temps. Lorsqu'un nouvel événement est publié sur un topic, il est en fait ajouté à l'une des partitions du topic. Les événements avec la même clé

d'événement (par exemple, un ID client ou un ID de véhicule) sont écrits dans la même partition, et Kafka garantit que tout consommateur d'une partition donnée lira toujours les événements de cette partition dans exactement le même ordre dans lequel ils ont été écrits.

Schéma :



Cet exemple de topic possède quatre partitions P1–P4. Deux clients producteurs différents publient, indépendamment l'un de l'autre, de nouveaux événements sur le topic en écrivant des événements sur le réseau vers les partitions du topic. Les événements avec la même clé (indiqués par leur couleur dans le schéma) sont écrits dans la même partition. Notez que les deux producteurs peuvent écrire dans la même partition si nécessaire.

Pour rendre vos données tolérantes aux pannes et hautement disponibles, chaque topic peut être répliqué, même à travers des régions géographiques ou des centres de données, de sorte qu'il y ait toujours plusieurs brokers ayant une copie des données. Un paramètre de production courant est un facteur de réplication de 3, c'est-à-dire qu'il y aura toujours trois copies de vos données. Cette réplication est effectuée au niveau des partitions de topic.

4 DESIGN

4.1 Motivation

La conception de Kafka a été guidée par le besoin de créer une plateforme unifiée capable de gérer tous les flux de données en temps réel qu'une grande entreprise pourrait avoir. Pour cela, nous avons pris en compte un ensemble varié de cas d'utilisation :

- **Haut Débit** : Kafka devait offrir un débit élevé pour supporter des flux d'événements volumineux, comme l'agrégation de journaux en temps réel.
- **Gestion des Retards de Données** : Le système devait pouvoir gérer de grandes accumulations de données pour supporter des chargements périodiques de données à partir de systèmes hors ligne.
- **Faible Latence** : Kafka devait également permettre une livraison à faible latence pour répondre aux besoins plus traditionnels de messagerie.
- **Traitement en Temps Réel** : Nous voulions supporter un traitement partitionné et distribué en temps réel de ces flux pour créer de nouveaux flux dérivés, ce qui a motivé notre modèle de partitionnement et de consommateur.
- **Tolérance aux Pannes** : Lorsque le flux est alimenté dans d'autres systèmes de données, il fallait garantir la tolérance aux pannes en cas de défaillance des machines.

Ces exigences ont conduit à une conception avec plusieurs éléments uniques, plus proches d'un journal de base de données que d'un système de messagerie traditionnel. Les sections suivantes détailleront certains éléments de cette conception.

4.2 Persistance

Utilisation du Système de Fichiers

Kafka utilise le système de fichiers pour stocker et mettre en cache les messages. Contrairement à la perception courante que "les disques sont lents", leur performance peut être étonnamment rapide lorsqu'ils sont utilisés correctement. En particulier, les écritures séquentielles sur disque sont très efficaces et optimisées par les systèmes d'exploitation modernes.

Performance des Disques

Il y a une différence significative entre les performances des écritures séquentielles et aléatoires sur disque :

- **Écritures Séquentielles** : Très rapides, pouvant atteindre environ 600 Mo/sec sur une configuration RAID-5 avec des disques SATA 7200rpm.
- **Écritures Aléatoires** : Beaucoup plus lentes, ne gérant qu'environ 100k écritures/sec.

Cette différence est due au fait que les écritures séquentielles sont prévisibles et peuvent être optimisées par le système d'exploitation, tandis que les écritures aléatoires ne le sont pas.

Utilisation de la Mémoire Cache

Les systèmes d'exploitation modernes utilisent intensivement la mémoire principale comme cache de disque. Cela signifie que :

- Toute la mémoire libre est utilisée pour mettre en cache les données du disque.
- Les données lues ou écrites passent par ce cache unifié, ce qui améliore les performances.

En utilisant le cache de disque du système d'exploitation (pagecache), Kafka peut stocker les données de manière plus compacte, ce qui :

- Double effectivement la mémoire disponible pour le cache.
- Évite les pénalités de performance liées à la gestion de la mémoire dans la JVM (Java Virtual Machine).

Conception Simplifiée

Plutôt que de conserver les données en mémoire et de les écrire sur le disque seulement lorsque la mémoire est pleine, Kafka :

- Écrit immédiatement toutes les données dans un journal persistant sur le système de fichiers.
- Utilise le pagecache du noyau pour gérer ces écritures de manière efficace.

Cela simplifie le code de Kafka car la gestion de la cohérence entre le cache et le système de fichiers est déléguée au système d'exploitation, qui le fait de manière plus efficace et fiable.

Avantages de cette Approche

1. **Cache Agrandi** : En utilisant la mémoire libre pour le cache de disque, on double la mémoire disponible pour le cache des données.
2. **Réduction des Pénalités de GC** : La collecte de déchets (GC) en Java devient moins problématique car les données sont stockées de manière compacte et en dehors de la JVM.
3. **Redémarrages Plus Rapides** : Le cache reste "chaud" même après un redémarrage, évitant ainsi les lenteurs liées à la reconstruction du cache en mémoire.
4. **Simplicité de Code** : La logique de gestion du cache est simplifiée, car elle est gérée par le système d'exploitation plutôt que par le processus Kafka lui-même.

Cette approche permet à Kafka de gérer efficacement de grandes quantités de données en temps réel, tout en offrant des performances élevées et une grande fiabilité.

Kafka mise sur l'efficacité pour gérer efficacement de grands volumes de données en temps réel. Pour y parvenir, deux principaux défis sont relevés :

1. **Réduction des Opérations d'E/S de Petite Taille** : Kafka utilise un modèle de "jeu de messages" qui regroupe plusieurs messages ensemble lors des opérations d'entrée/sortie. Cela minimise le surcoût de latence associé aux opérations individuelles et améliore les performances globales.
2. **Optimisation des Copies de Données** : En utilisant un format de message binaire standardisé, Kafka évite les copies redondantes de données entre les producteurs, les courtiers et les consommateurs. Cela permet un transfert de données plus efficace et des performances accrues.

En combinant ces techniques avec l'utilisation du cache de page et de l'appel système sendfile, Kafka peut fournir des performances optimales, même dans des environnements à haute charge. Cette approche permet à Kafka de répondre efficacement aux besoins des applications à forte intensité de données en temps réel.

Dans Kafka, le producteur (producer) et le consommateur (consumer) jouent des rôles cruciaux dans le traitement efficace des flux de données. Voici un aperçu de leur fonctionnement :

4.4 Le Producteur

Équilibrage de Charge : Le producteur envoie directement les données au courtier (broker) qui est le leader pour la partition sans aucune intervention de routage. Les nœuds Kafka peuvent fournir des métadonnées sur les serveurs actifs et les leaders des partitions à tout moment, permettant ainsi au producteur de diriger ses requêtes de manière appropriée.

Contrôle de la Partition : Le client contrôle vers quelle partition il publie les messages. Cela peut être fait de manière aléatoire ou en utilisant une fonction de partitionnement sémantique. Par exemple, si une clé utilisateur est choisie, toutes les données pour un utilisateur donné seront envoyées à la même partition. Cela permet aux consommateurs de faire des hypothèses de localité sur leur consommation.

Envoi Asynchrone : Le producteur tente d'accumuler les données en mémoire et d'envoyer de plus gros lots dans une seule requête pour optimiser le regroupement (batching). Cette accumulation peut être configurée pour accumuler un nombre fixe de messages ou attendre une latence fixe. Cela permet de réduire le nombre d'opérations d'entrée/sortie sur les serveurs et d'améliorer le débit.

4.5 Le Consommateur

Le consommateur envoie des requêtes "fetch" aux courtiers qui dirigent les partitions qu'il souhaite consommer. Il spécifie son décalage dans le journal avec chaque demande et reçoit un morceau de journal à partir de cette position. Cela donne au consommateur un contrôle significatif sur cette position et lui permet de rembobiner pour reconsommer des données si nécessaire.

En utilisant ces mécanismes, Kafka assure une gestion efficace des flux de données, offrant une haute performance et une grande flexibilité pour répondre aux besoins des applications distribuées à grande échelle.

4.6 Sémantique de la Livraison des Messages

Au Plus Une Fois : Les messages peuvent être perdus mais ne sont jamais renvoyés.

Au Moins Une Fois : Les messages ne sont jamais perdus mais peuvent être renvoyés.

Exactement Une Fois : Chaque message est livré une seule fois et exactement une fois.

Kafka propose des garanties de livraison de messages directes. Lors de la publication d'un message, il est "commis" dans le journal (log). Une fois qu'un message est commis, il ne sera pas perdu tant qu'au moins un courtier (broker) répliquant la partition à laquelle ce message a été écrit reste "actif". Cela garantit que les messages ne seront pas perdus, même en cas de panne d'un courtier.

Avant la version 0.11.0.0, si un producteur ne recevait pas de réponse indiquant qu'un message était commis, il devait généralement renvoyer le message, ce qui garantissait au moins une fois la livraison des messages. Depuis la version 0.11.0.0, le producteur Kafka prend également en charge une option de livraison idempotente, garantissant que le renvoi ne génère pas d'entrées en double dans le journal.

 Concrètement, chaque message envoyé par le producteur est associé à un numéro de séquence unique. Ce numéro de séquence est envoyé avec le message et est utilisé par le courtier (broker) pour garantir qu'un message avec le même numéro de séquence ne sera pas écrit deux fois dans le journal, même en cas de renvoi du message par le producteur en raison d'une erreur. Cela permet d'éviter les doublons dans le journal, assurant ainsi une livraison fiable des messages sans duplication.

Pour le consommateur, le positionnement dans le journal est contrôlé par le consommateur lui-même. Si le consommateur échoue, un nouveau processus doit choisir une position appropriée à partir de laquelle commencer le traitement. Les options de traitement offrent des garanties d'au moins une fois ou d'au plus une fois selon le moment où la position est enregistrée.

Pour les traitements nécessitant une garantie d'exactement une fois, Kafka propose des capacités de producteur et de consommateur transactionnels. Cela permet de coordonner le positionnement du consommateur avec les données traitées dans un flux transactionnel.

Kafka fournit donc une gamme complète de garanties de livraison des messages, des plus simples aux plus robustes, permettant aux développeurs de choisir en fonction des besoins spécifiques de leurs applications.

4.7 Replication

Kafka réplique les journaux pour chaque partition de topic sur un nombre configurable de serveurs, permettant ainsi une bascule automatique vers ces répliques en cas de défaillance d'un serveur, maintenant ainsi la disponibilité des messages.

La réplication est essentielle dans Kafka et est activée par défaut, avec la possibilité de configurer le facteur de réplication sur une base de topic.

Chaque partition de Kafka a un leader et des followers. Les écritures vont au leader, tandis que les lectures peuvent être effectuées sur le leader ou les followers. Les journaux des followers sont identiques à ceux du leader.

La viabilité des courtiers est déterminée par deux conditions : maintenir une session active avec le contrôleur et répliquer les écritures du leader. Les nœuds "en synchronisation" sont ceux qui satisfont ces deux conditions.

Kafka assure qu'un message engagé ne sera pas perdu tant qu'il y aura au moins une réplique en synchronisation en vie.

Kafka reste disponible en cas de défaillance des nœuds après une courte période de basculement, mais peut ne pas rester disponible en cas de partitions réseau.

lorsqu'on dit que Kafka peut ne pas rester disponible en cas de partitions réseau, cela signifie que dans des situations où des parties du réseau ne peuvent pas communiquer entre elles, Kafka peut rencontrer des problèmes de disponibilité.

Une partition réseau se produit lorsque les nœuds d'un cluster Kafka sont divisés en groupes isolés en raison de problèmes de connectivité réseau. Par exemple, si un segment du réseau échoue, les nœuds situés de part et d'autre de cette défaillance peuvent ne plus être en mesure de se communiquer.

Dans de telles situations, Kafka peut ne pas être en mesure de garantir que tous les messages seront accessibles ou que les opérations de lecture et d'écriture seront effectuées de manière cohérente. Cela peut entraîner des retards dans la réplication des messages, des incohérences dans les données ou même une perte de disponibilité totale du système pour certaines parties du cluster.

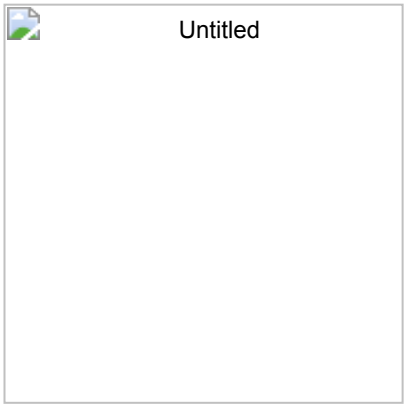
Disponibilité et durabilité garantie

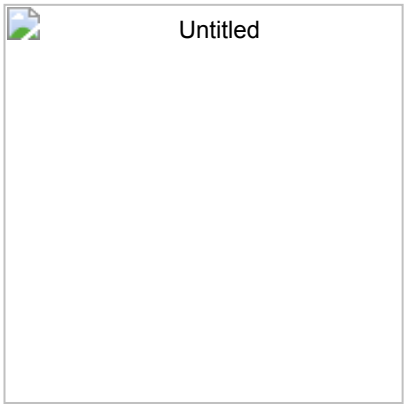
1. **Choix des répliques pour les producteurs** : Les producteurs peuvent choisir d'attendre que le message soit confirmé par 0, 1 ou toutes les répliques (-1). Par défaut, lorsque acks = all, l'approbation se produit dès que toutes les répliques en synchronisation actuelles ont reçu le message. Cela garantit la disponibilité maximale de

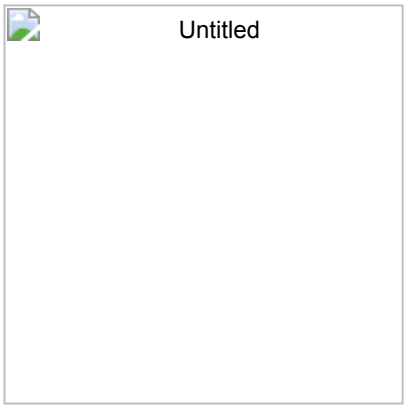
la partition, mais peut entraîner la perte de messages si toutes les répliques en synchronisation ne reçoivent pas le message. Pour les utilisateurs qui préfèrent la durabilité à la disponibilité, Kafka propose deux configurations au niveau du sujet :

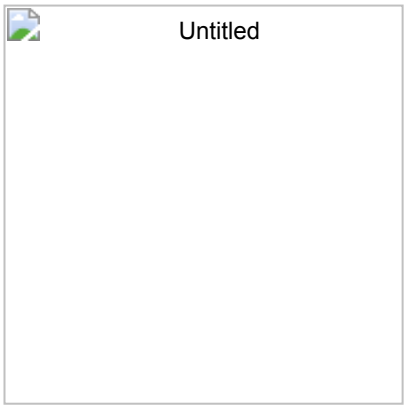
- **Désactiver l'élection de leader non fiable** : Si toutes les répliques deviennent indisponibles, la partition restera indisponible jusqu'à ce que le leader le plus récent redevienne disponible. Cela préfère l'indisponibilité au risque de perte de messages.
- **Spécifier une taille ISR minimale** : La partition n'acceptera les écritures que si la taille de l'ISR est supérieure à un minimum spécifié, pour éviter la perte de messages écrits sur une seule réplique devenue indisponible. Cela offre un compromis entre la cohérence et la disponibilité.

2. **Gestion des répliques** : Un cluster Kafka gère des centaines ou des milliers de partitions. Il tente d'équilibrer les partitions dans le cluster de manière équitable et d'optimiser le processus d'élection de leader pour minimiser les périodes d'indisponibilité. Le rôle spécial du "contrôleur" dans le cluster est de gérer l'enregistrement des courtiers et de coordonner les élections de leader en cas de défaillance d'un nœud. Cela permet de réduire les coûts et la durée des élections de leader pour un grand nombre de partitions. Si le contrôleur lui-même échoue, un autre contrôleur est élu pour prendre sa place






Untitled


Untitled