Le papier de Leslie Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System" (1978), aborde le problème fondamental de la synchronisation dans les systèmes distribués. Dans ce contexte, un système distribué est composé de plusieurs processus indépendants s'exécutant sur différentes machines, sans horloge physique commune pour les synchroniser.

Le problème soulevé est qu'il est difficile de déterminer l'ordre exact des événements (tels que l'envoi et la réception de messages) dans ces systèmes distribués, car chaque processus dispose de sa propre horloge, qui peut ne pas être synchronisée avec les autres. Cela rend la notion de temps ambigüe et pose des défis pour la cohérence de l'état global du système.

Les enjeux associés incluent la nécessité de coordonner les activités entre les processus de manière cohérente, d'éviter les incohérences dans les bases de données réparties, et de garantir que les opérations sur les ressources partagées se déroulent de manière ordonnée. Lamport propose une solution basée sur l'utilisation de "horloges logiques" pour ordonner les événements et fournir une méthode fiable de causalité dans les systèmes distribués, jetant ainsi les bases pour le développement de protocoles de consensus et d'algorithmes de tolérance aux pannes.

La proposition donnée est :

> Soit aaa et bbb dans NNN, si h(a)<h(b)h(a) < h(b)h(a)<h(b) équivaut à a→ba \rightarrow ba→b, cela implique que la relation →\rightarrow→ est d'ordre total, réflexive, transitive, et antisymétrique.

Analysons cela en détail.

1. Relation →\rightarrow→ dans les systèmes distribués

Dans le contexte du papier de Lamport, la relation a→ba \rightarrow ba→b signifie que l'événement aaa précède causalement l'événement bbb dans un système distribué. Il s'agit d'une relation de causalité, où a→ba \rightarrow ba→b indique que aaa peut influencer bbb.

2. Propriétés de la relation d'ordre

Pour qu'une relation soit considérée comme un ordre total, réflexif, transitive et antisymétrique, elle doit satisfaire les critères suivants :

- **Réflexivité** : Pour tout aaa, a→aa \rightarrow aa→a.
- **Transitivité** : Si a→ba \rightarrow ba→b et b→cb \rightarrow cb→c, alors a→ca \rightarrow ca→c.
- **Antisymétrie** : Si a→ba \rightarrow ba→b et b→ab \rightarrow ab→a, alors a=ba = ba=b.
- **Totalité** : Pour tous les aaa et bbb, soit a→ba \rightarrow ba→b soit b→ab \rightarrow ab→a.

3. Vérification de la proposition

Réflexivité et transitivité

Dans le modèle de Lamport, la relation →\rightarrow→ est effectivement définie comme réflexive (un événement précède lui-même) et transitive (la causalité se propage). Ces propriétés sont donc satisfaites.

Antisymétrie

L'antisymétrie n'est pas garantie dans un système distribué. En effet, il peut exister des événements aaa et bbb tels que ni a→ba \rightarrow ba→b ni b→ab \rightarrow ab→a ne soient vrais, ce qui signifie qu'ils sont en concurrence (c'est-à-dire indépendants l'un de l'autre). La relation →\rightarrow→ n'est donc pas antisymétrique en général.

Totalité

La relation de causalité →\rightarrow→ n'est pas un ordre total, car il est possible d'avoir deux événements indépendants aaa et bbb tels que ni a→ba \rightarrow ba→b ni b→ab \rightarrow ab→a ne soient vrais. Ainsi, →\rightarrow→ n'est pas total, mais partiel.

Conclusion

La proposition selon laquelle la relation →\rightarrow→ serait un ordre total, réflexif, transitive, et antisymétrique n'est pas correcte. La relation →\rightarrow→ est en réalité un ordre partiel, réflexif et transitive, mais elle n'est ni totale ni antisymétrique.

Notion de concurrence d’évènements qui empêche l’antisymétrie.

La **causalité** est un concept fondamental en informatique, physique, philosophie, et dans d'autres disciplines, qui exprime une relation de cause à effet entre deux événements. Lorsqu'on dit qu'un événement AAA **cause** un événement BBB, cela signifie que l'occurrence de AAA influence, déclenche ou rend possible l'occurrence de BBB. En d'autres termes, AAA précède BBB dans le temps, et il y a un lien logique ou physique entre les deux.

Causalité dans les systèmes distribués

Dans le contexte des systèmes distribués, la causalité est un concept clé pour ordonner les événements dans un environnement où il n'y a pas d'horloge globale commune. Leslie Lamport a introduit l'idée de **causalité logique** pour fournir une manière de déterminer l'ordre des événements à l'aide de la relation →\rightarrow→, qui indique que :

- a→ba \rightarrow ba→b signifie que l'événement aaa précède causalement l'événement bbb, c'est-à-dire que l'existence ou le résultat de aaa peut influencer bbb. Cette relation peut être établie de plusieurs manières :

    - **Ordre local** : Si aaa et bbb sont des événements au sein du même processus et aaa survient avant bbb, alors a→ba \rightarrow ba→b.
    - **Envoi et réception de messages** : Si un événement aaa correspond à l'envoi d'un message et que bbb est la réception de ce message dans un autre processus, alors a→ba \rightarrow ba→b.
    - **Transitivité** : Si a→ba \rightarrow ba→b et b→cb \rightarrow cb→c, alors a→ca \rightarrow ca→c.

Causalité et horloges logiques

Pour représenter la causalité dans les systèmes distribués, Lamport a proposé l'utilisation d'**horloges logiques**, qui associent des horodatages aux événements. Ces horodatages permettent de comparer les événements et de déterminer si un événement précède causalement un autre. Si h(a)<h(b)h(a) < h(b)h(a)<h(b), cela implique que a→ba \rightarrow ba→b.

Cependant, les horloges logiques ne capturent pas toutes les relations de causalité possibles. Parfois, deux événements peuvent être **concurrents**, ce qui signifie qu'il n'existe aucun lien de causalité entre eux, et donc ni a→ba \rightarrow ba→b ni b→ab \rightarrow ab→a ne sont vrais.

Importance de la causalité

La notion de causalité est cruciale pour de nombreux aspects des systèmes distribués, notamment :

- **Synchronisation** : Garantir que les actions dans un système se produisent dans un ordre cohérent.
- **Cohérence** : Maintenir l'intégrité des données dans des bases de données réparties.
- **Tolérance aux pannes** : Assurer la continuité du service même en cas de défaillance de certains composants.

En résumé, la causalité aide à ordonner les événements dans un système distribué en l'absence d'une horloge physique globale, fournissant ainsi une base pour comprendre et gérer les interactions entre les différents processus.

Dans les systèmes distribués, comme ceux abordés par Leslie Lamport dans son papier sur les horloges logiques, la relation d'antisymétrie n'est pas toujours conservée à cause de l'absence d'une horloge globale. Pour pallier cette limitation, Lamport propose d'utiliser des **compteurs d'entiers** (ou **horloges logiques**) qui possèdent certaines caractéristiques pour établir un ordre des événements.

Caractéristiques des compteurs d'entiers dans le modèle de Lamport

1. **Incrémentation locale** :
    - Chaque processus dans le système maintient un compteur d'entier local qui s'incrémente à chaque événement interne. Ainsi, chaque fois qu'un

événement se produit (comme l'envoi ou la réception d'un message), le compteur est augmenté de 1.

2. **Envoi de messages** :
    - Lorsqu'un processus envoie un message, il joint la valeur actuelle de son compteur à ce message. Cela permet à l'événement d'envoi d'être horodaté par la valeur du compteur au moment de l'envoi.
3. **Réception de messages** :
    - À la réception d'un message, le processus qui reçoit le message met à jour son propre compteur. Il prend le maximum entre sa valeur actuelle et la valeur du compteur reçue dans le message, puis il incrémente cette valeur. Cela garantit que le compteur reflète les événements passés de manière cohérente.
    - Formule : Creceiving=max⁡(Creceiving,Csending)+1C_{receiving} = \max(C_{receiving}, C_{sending}) + 1Creceiving=max(Creceiving,Csending)+1
4. où CreceivingC_{receiving}Creceiving est le compteur du processus qui reçoit le message, et CsendingC_{sending}Csending est le compteur du processus qui a envoyé le message.
5. **Relation de causalité** :
    - Grâce à ces compteurs, on peut établir une relation de causalité entre les événements. Si un événement aaa précède causalement un événement bbb (c'est-à-dire a→ba \rightarrow ba→b), alors l'horodatage associé à aaa (la valeur du compteur) sera strictement inférieur à celui de bbb : h(a)<h(b)h(a) < h(b)h(a)<h(b)
6. **Cohérence et ordre partiel** :
    - Les compteurs d'entiers permettent de déterminer un ordre partiel des événements dans le système. Bien que cela ne résolve pas toutes les ambiguïtés causales (car il peut y avoir des événements concurrents), cela fournit une méthode systématique pour ordonner les événements causaux.

Avantages de cette approche

- **Évitement de l'horloge globale** : Les compteurs d'entiers permettent d'éviter la nécessité d'une horloge globale tout en fournissant un mécanisme pour ordonner les événements.
- **Simplicité** : L'utilisation d'entiers pour représenter le temps et les relations de causalité est conceptuellement simple et facile à mettre en œuvre dans des systèmes distribués.
- **Scalabilité** : Ce modèle est scalable, car chaque processus gère son propre compteur sans nécessiter de coordination avec d'autres processus.

Conclusion

Lamport a proposé l'utilisation de compteurs d'entiers pour gérer la causalité dans les systèmes distribués, remplaçant ainsi l'absence d'une fonction d'horodatage globale. Ces compteurs permettent d'établir une relation causale et d'ordonner les événements de manière efficace, tout en respectant les caractéristiques fondamentales de la causalité.

'utilisation d'une matrice pour gérer les messages dans un système distribué présente plusieurs problèmes et défis associés. Voici les principaux problèmes à considérer :

1. Scalabilité

- **Problème** : À mesure que le nombre de processus dans le système augmente, la taille de la matrice augmente de manière quadratique, ce qui peut poser des problèmes en termes de mémoire et de performances.
- **Conséquence** : Dans un grand système distribué, la matrice peut devenir trop volumineuse pour être stockée et manipulée efficacement, rendant difficile le suivi des messages et des dépendances.

2. Mise à jour et incohérences

- **Problème** : Les mises à jour de la matrice doivent être effectuées avec soin pour éviter des incohérences, surtout lorsque plusieurs processus interagissent simultanément.
- **Conséquence** : Si une mise à jour n'est pas correctement synchronisée, cela peut entraîner des erreurs dans la gestion des dépendances, permettant à un processus de traiter un message alors que ses pré-requis n'ont pas été remplis.

3. Latence

- **Problème** : L'introduction d'une matrice de dépendance peut ajouter une latence supplémentaire à la communication entre processus, en particulier lors de la vérification des dépendances avant le traitement des messages.
- **Conséquence** : Cela peut ralentir le traitement global des messages, ce qui n'est pas acceptable dans des applications où la réactivité est essentielle.

4. Complexité de la logique de traitement

- **Problème** : L'implémentation de la logique de traitement des messages à l'aide d'une matrice peut devenir complexe, notamment en ce qui concerne la gestion des dépendances et la vérification des messages reçus.
- **Conséquence** : Cette complexité accrue peut entraîner des bogues dans le code, rendant le système moins fiable et plus difficile à maintenir.

2. File d'attente FIFO

Objectif

La **file d'attente FIFO** est utilisée pour gérer l'ordre de traitement des transactions. Cette méthode garantit que les transactions sont traitées dans l'ordre dans lequel elles ont été reçues, ce qui est essentiel pour préserver l'intégrité des transactions dans le système.

Fonctionnement

- **Ajout à la file** : Lorsqu'une transaction arrive, elle est ajoutée à la fin de la file d'attente FIFO.
- **Traitement** : Les transactions sont traitées dans l'ordre où elles ont été ajoutées à la file d'attente. Cela garantit que si une transaction doit être traitée avant une autre (selon la matrice de dépendance), la première transaction est traitée avant la seconde.

3. Intégration de la matrice et de la file FIFO

Pour gérer l'ordre des transactions de manière efficace, la matrice de dépendance et la file FIFO peuvent être intégrées :

1. **Vérification des dépendances** : Avant de traiter une transaction, le système consulte la matrice de dépendance pour vérifier si d'autres transactions doivent être traitées en premier.
2. **Mise à jour de la file FIFO** :
    - Si une transaction a des dépendances non satisfaites, elle est mise en attente dans la file FIFO jusqu'à ce que toutes les transactions dont elle dépend soient traitées.
    - Lorsqu'une transaction est traitée, le système met à jour la matrice pour refléter que certaines dépendances peuvent désormais être satisfaites.

Synchronisation Temporelle

- **Temps Universel Coordiné (UTC)** : Le GPS repose sur une mesure extrêmement précise du temps, qui est essentielle pour calculer les distances entre les satellites et les récepteurs GPS.
- **Impact** : Le GPS utilise des horloges atomiques de haute précision dans ses satellites, ce qui permet de synchroniser le temps globalement. Cette synchronisation est cruciale pour garantir que les signaux GPS envoyés par les satellites arrivent au récepteur dans le bon ordre et au bon moment.

Temps Absolu : Fournit une mesure uniforme et cohérente du temps à travers le monde, ce qui est essentiel pour des applications nécessitant une précision de localisation, comme le GPS.

Synchronisation NTP (Network Time Protocol)

- **Description** : Pour maintenir une heure précise, les ordinateurs se synchronisent souvent avec des serveurs de temps via le protocole NTP (Network Time Protocol).
- **Fonction** : NTP ajuste l'horloge système de l'ordinateur en se basant sur des horloges atomiques sur Internet, assurant que tous les ordinateurs d'un réseau partagent une heure cohérente et précise.