Les horloges logiques

Yann Letourneur, Yohan Chevrier-Pivot
1er décembre 2025

Résumé

Les horloges logiques constituent un mécanisme fondamental pour ordonner les événements dans les systèmes distribués, où il n'existe pas de notion globale du temps physique. Introduites par Leslie Lamport en 1978, elles permettent de déterminer la causalité entre événements et d'assurer la cohérence des données. Trois types d'horloges coexistent : l'horloge linéaire de Lamport, qui offre un ordre total simple mais ne capture pas tous les liens causaux ; l'horloge vectorielle de Fidge et Mattern, qui détecte la concurrence entre événements ; et l'horloge matricielle de Wuu et Bernstein, qui ajoute la connaissance mutuelle entre processus. Ces mécanismes sont essentiels pour la synchronisation, le débogage et le suivi dans les architectures distribuées modernes.

Introduction

Dans les systèmes distribués, plusieurs processus s'exécutent simultanément sur différentes machines sans synchronisation d'horloge physique globale. Cette absence de référence temporelle commune pose un défi majeur : comment déterminer l'ordre dans lequel les événements se produisent ? Selon Leslie Lamport (1978), "la nature logique du temps est d'une importance primordiale lors de la conception ou de l'analyse de systèmes distribués". Sans mécanisme d'ordonnancement, il devient impossible de garantir la cohérence des données, d'assurer une synchronisation correcte ou de déboguer efficacement un système.

Les horloges logiques répondent à ce besoin en établissant un ordre partiel ou total sur les événements, indépendamment du temps physique. Elles reposent sur la notion de causalité : si un événement e_1 influence potentiellement un événement e_2, alors e_1 doit être considéré comme antérieur à e_2. Cette relation de dépendance causale, notée $e_1 \rightarrow e_2$, structure l'ensemble des événements du système. Les horloges logiques permettent ainsi de tracer le cône de causalité de chaque événement, c'est-à-dire l'ensemble des événements qui l'ont précédé et influencé.

Événements et causalité dans les systèmes distribués

Un système distribué se compose de n processus communiquant via des canaux bidirectionnels FIFO (First In, First Out) et sans perte. Chaque processus peut effectuer trois types d'événements : les calculs internes, l'envoi de messages et la réception de messages. La relation de causalité $e_1 \rightarrow e_2$ signifie que "e_1 se produit avant e_2" selon un ordre partiel : tous les événements ne sont pas nécessairement comparables entre eux.

Cette relation causale définit le cône de causalité d'un événement e, qui regroupe tous les événements ayant directement ou indirectement précédé e. Capturer cette causalité est essentiel pour maintenir la cohérence dans un système où les processus évoluent de manière asynchrone.

Horloge logique linéaire (Lamport, 1978)

L'horloge de Lamport représente le temps logique par un simple entier. Chaque processus P_i maintient sa propre horloge logique h_i, qui reflète sa vue locale du temps global. Deux règles régissent son fonctionnement :

Règle R1 : Avant chaque événement, le processus incrémente son horloge locale de d (généralement d = 1) :
$h_i := h_i + d$

Règle R2 : Lors de la réception d'un message (m, h), le processus met à jour son horloge avec le maximum entre sa valeur actuelle et celle reçue, puis applique R1 avant de délivrer le message :
$h_i := \max(h_i, h) + d$

Cette approche garantit la propriété de monotonie : si $e_1 \rightarrow e_2$, alors $h(e_1) < h(e_2)$. Cependant, la réciproque n'est pas vraie : deux événements avec $h(e_1) < h(e_2)$ ne sont pas nécessairement en relation causale. Pour obtenir un ordre total, il suffit d'ajouter l'identifiant du processus émetteur : e_1 est avant e_2 si et seulement si ($h_1 < h_2$) ou ($h_1 = h_2$ et $P_1 < P_2$).

Exemple : Édition collaborative d'un document

Trois utilisateurs (P_1, P_2, P_3) éditent un document partagé. État initial : $h_1 = h_2 = h_3 = 0$.

- P_1 ajoute "Bonjour" $\rightarrow h_1 = 1$
- P_1 envoie sa modification à P_2 (message avec h=1)
- P_2 effectue un calcul interne $\rightarrow h_2 = 1$
- P_2 reçoit le message de $P_1 \rightarrow h_2 = \max(1, 1) + 1 = 2$
- P_3 ajoute "monde" $\rightarrow h_3 = 1$
- P_2 envoie sa modification à P_3 (message avec h=2)
- P_3 reçoit le message de $P_2 \rightarrow h_3 = \max(1, 2) + 1 = 3$

On peut ordonner les événements, mais on ne sait pas si "Bonjour" et "monde" ont été écrits indépendamment ou si l'un a influencé l'autre.

Avantages : L'horloge de Lamport se distingue par sa simplicité d'implémentation et son efficacité en termes d'espace mémoire (O(1) par processus) et de communication (un seul entier transmis par message). Elle fournit un ordre total sur les événements, ce qui est suffisant pour de nombreuses applications comme les algorithmes d'exclusion mutuelle ou l'estampillage de transactions.

Inconvénients : L'horloge de Lamport ne permet pas de détecter la concurrence entre événements indépendants. Si $h(e_1) < h(e_2)$, on ne peut pas déterminer si e_1 a réellement causé e_2 ou s'ils sont simplement concurrents. Cette limitation rend l'horloge de Lamport inadaptée pour les systèmes nécessitant une détection précise de la causalité, comme les bases de données distribuées avec résolution de conflits ou les systèmes de versionning.

Horloge logique vectorielle (Fidge & Mattern, années 1980)

L'horloge vectorielle représente le temps par un vecteur de taille n, où n est le nombre de processus dans le système. Chaque processus P_i possède un vecteur v_i, où $v_i[k]$ représente la connaissance qu'a P_i du temps local de P□. Les règles de mise à jour sont les suivantes :

Règle R1 : Avant de produire un événement, le processus incrémente son propre compteur de d :
$v_i[i] := v_i[i] + d$

Règle R2 : Lors de la réception d'un message (m, v), le processus met à jour toutes les composantes de son vecteur avec le maximum des valeurs présentes et reçues, puis applique R1 :
$\forall k \in [1, n] : v_i[k] := \max(v_i[k], v[k]) + d$

Les horloges vectorielles permettent de caractériser précisément la causalité :

- **Événements liés** : $e_1 \rightarrow e_2 \Leftrightarrow v_1 < v_2 \Leftrightarrow \forall k : v_1[k] \leq v_2[k]$ et $\exists k : v_1[k] < v_2[k]$
- **Événements concurrents** : $e_1 \parallel e_2 \Leftrightarrow v_1 \not< v_2$ et $v_1 \not> v_2$

Exemple : Édition collaborative avec détection de conflits

Reprenons les trois utilisateurs (P_1, P_2, P_3). État initial : $v_1 = [0,0,0]$, $v_2 = [0,0,0]$, $v_3 = [0,0,0]$.

- P_1 ajoute "Bonjour" → $v_1 = [1,0,0]$
- P_1 envoie à P_2 (message avec v=[1,0,0])
- P_2 fait un calcul → $v_2 = [0,1,0]$
- P_2 reçoit de P_1 → $v_2 = [\max(0,1), \max(1,0), \max(0,0)] + [0,1,0] = [1,2,0]$
- P_3 ajoute "monde" → $v_3 = [0,0,1]$
- P_2 envoie à P_3 (message avec v=[1,2,0])
- P_3 reçoit de P_2 → $v_3 = [1,2,2]$

Maintenant, on peut détecter que "Bonjour" ([1,0,0]) et "monde" initial ([0,0,1]) sont concurrents car $[1,0,0] \not< [0,0,1]$ et $[1,0,0] \not> [0,0,1]$. Le système peut alors demander à l'utilisateur de résoudre le conflit.

Avantages : L'horloge vectorielle capture précisément les relations causales entre événements et permet de détecter efficacement la concurrence. Cette propriété est essentielle pour les systèmes de réplication optimiste, les bases de données distribuées (comme Riak ou Cassandra), et les algorithmes de détection de points de reprise cohérents. Elle offre une caractérisation complète de la causalité : $e_1 \rightarrow e_2$ si et seulement si $v_1 < v_2$.

Inconvénients : L'horloge vectorielle nécessite un espace mémoire de O(n) par processus et transmet un vecteur de taille n avec chaque message, ce qui augmente significativement le coût en communication par rapport à l'horloge de Lamport. Pour les systèmes avec un grand nombre de processus, cette surcharge peut devenir prohibitive. De plus, la taille des vecteurs est fixée par le nombre de processus dans le système, ce qui complique la gestion des systèmes dynamiques où des processus peuvent rejoindre ou quitter le système.

Horloge logique matricielle (Wuu & Bernstein, 1984)

L'horloge matricielle étend le concept d'horloge vectorielle en représentant le temps par une matrice n × n. Chaque processus P_i possède une matrice M_i, où $M_i[k, l]$ correspond à la vue qu'à P_i des connaissances de P□ sur l'horloge logique de P□. La ligne $M_i[i, :]$ fonctionne exactement comme un vecteur d'horloge vectorielle.

Les règles de mise à jour sont :

Règle R1 : Avant chaque événement, le processus incrémente son horloge locale de d :
$M_i[i, i] := M_i[i, i] + d$

Règle R2 : À la réception d'un message accompagné de la matrice de l'émetteur M□, le processus met à jour ses connaissances :
$\forall k, l \in [1, n] : M_i[k, l] := \max(M_i[k, l], M□[k, l])$

L'horloge matricielle conserve toutes les propriétés de l'horloge vectorielle, mais ajoute une propriété cruciale de connaissance mutuelle :
$\min□(M_i[k, i]) \geq t \Rightarrow$ Le processus P_i sait que tous les autres processus connaissent ses progrès jusqu'à son temps t

Exemple : Validation distribuée d'une transaction

Trois serveurs bancaires (P_1, P_2, P_3) doivent valider une transaction. P_1 veut s'assurer que tous ont bien reçu sa demande avant de finaliser.

État initial : $M_1 = M_2 = M_3 = [[0,0,0], [0,0,0], [0,0,0]]$

- P_1 initie la transaction → $M_1[1,1] = 1$, donc $M_1 = [[1,0,0], [0,0,0], [0,0,0]]$
- P_1 envoie à P_2 (avec M_1)
- P_2 reçoit → $M_2 = [[1,0,0], [1,1,0], [0,0,0]]$ (P_2 sait que P_1 a progressé et met à jour sa ligne)
- P_2 envoie à P_3 (avec M_2)
- P_3 reçoit → $M_3 = [[1,0,0], [1,1,0], [1,1,1]]$ (P_3 connaît l'état de tous)
- P_3 envoie à P_1 (avec M_3)
- P_1 reçoit → $M_1 = [[1,1,1], [1,1,0], [1,1,1]]$

P_1 calcule $\min(M_1[1,1], M_1[2,1], M_1[3,1]) = \min(1, 1, 1) = 1$. P_1 sait maintenant que tous les processus ont connaissance de sa transaction initiale et peut procéder à la validation finale en toute sécurité.

Avantages : L'horloge matricielle permet de raisonner sur la connaissance mutuelle entre processus, une propriété indispensable pour certains protocoles distribués avancés. Elle est particulièrement utile pour les algorithmes de terminaison distribuée, où un processus doit s'assurer que tous les autres ont atteint un certain état, les protocoles de validation en deux phases (2PC), le garbage collection distribué, et la détection de la stabilité globale. Elle fournit des informations que ni l'horloge de Lamport ni l'horloge vectorielle ne peuvent offrir.

Inconvénients : L'horloge matricielle souffre d'une complexité spatiale et communicationnelle très élevée. Chaque processus doit maintenir une matrice de taille $O(n^2)$, et chaque message transporte cette matrice complète. Pour des systèmes avec un nombre important de processus, cette surcharge devient rapidement prohibitive et limite l'utilisation pratique des horloges matricielles aux systèmes de petite à moyenne taille. De plus, la complexité de gestion et d'implémentation est significativement plus élevée que pour les horloges vectorielles.

Conclusion

Les horloges logiques offrent des mécanismes adaptés à différents besoins dans les systèmes distribués. L'horloge de Lamport fournit une solution simple et efficace pour établir un ordre total sur les événements, au prix d'une incapacité à détecter la concurrence. L'horloge vectorielle, plus complexe, capture précisément les relations causales et identifie les événements concurrents, ce qui en fait un choix privilégié pour les systèmes de réplication et de cohérence. Enfin, l'horloge matricielle ajoute une dimension de connaissance mutuelle indispensable pour certains protocoles avancés. Le choix entre ces trois approches dépend du compromis entre précision sémantique et coût en espace mémoire et en communication : $O(1)$ pour Lamport, $O(n)$ pour les vecteurs et $O(n^2)$ pour les matrices.

Références commentées

• **Lamport (1978).** Article fondateur introduisant le concept d'horloge logique linéaire et la relation "happened-before", posant les bases de l'ordonnancement causal dans les systèmes distribués.

• **Fidge (1988) et Mattern (1988).** Ces travaux indépendants ont introduit simultanément les horloges vectorielles, permettant de caractériser précisément la causalité et de détecter la concurrence entre événements.

• **Wuu & Bernstein (1984).** Extension des horloges vectorielles vers les horloges matricielles, ajoutant la notion de connaissance mutuelle entre processus pour des protocoles distribués avancés.

Références

[1] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[2] Colin J. Fidge. Timestamps in message-passing systems that preserve the partial ordering. *Proceedings of the 11th Australian Computer Science Conference*, pages 56–66, 1988.

[3] Friedemann Mattern. Virtual time and global states of distributed systems. *Parallel and Distributed Algorithms*, pages 215–226, 1988.

[4] Gene T. J. Wuu and Arthur J. Bernstein. Efficient solutions to the replicated log and dictionary problems. *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, pages 233–242, 1984.