Prise d'instantané (snapshot) distribuée

Stéphane Devismes

Résumé

Le concept de prise d'instantané distribuée a été introduit par Chandy et Lamport en 1985 [CL85]. Le but est de déterminer une configuration du système à l'aide d'un algorithme distribué.

Un algorithme réalisant une prise d'instantané permet de résoudre une classe importante de problèmes : « la détection de propriétés stables ». Comme son nom l'indique, une *propriété stable* est une propriété qui reste vraie pour toujours à partir du moment où elle est vérifiée. Par exemple, « un calcul est terminé », « le système est interbloqué » (deadlock) ou encore « tous les jetons circulant dans le réseau ont disparu » sont des exemples de propriété stable. Bien entendu, le problème de la détection de propriétés stables consiste à concevoir un algorithme capable de détecter si une propriété stable donnée est vérifiée.

La prise d'instantané a d'autres applications, par exemple elle peut être utilisée pour faire des *points de contrôle* (checkpointing).

Table des matières

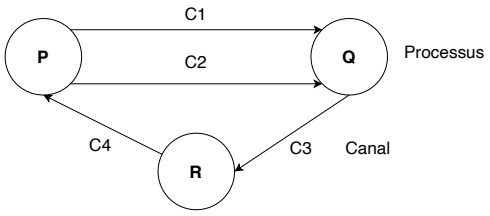

FIGURE 1 – Un système distribué avec trois processus et quatre canaux.

1 Introduction

Sur la difficulté de la prise d'instantané. Dans le modèle à passage de messages, un processus peut uniquement enregistrer son état ainsi que les messages qu'il envoie ou reçoit. Pour déterminer une configuration du système, un processus doit, en particulier, collecter l'ensemble des états des autres processus. Or, premier problème : puisque le système est asynchrone et que les processus n'ont pas accès à une horloge globale, on ne peut garantir qu'ils enregistrent leurs états locaux exactement au même moment. Ainsi, le problème est de collecter les états locaux et les états des canaux de communication de telle manière qu'ils permettent de reconstituer une configuration cohérente du système.

De plus, l'algorithme de prise d'instantané doit s'exécuter concuremment avec le système, et donc un algorithme sous-jacent, dont il cherche à évaluer la configuration. L'algorithme de prise d'instantané ne doit pas modifier l'exécution de ce dernier (il faut éviter le problème du chat de Schrödinger).

Allégorie. Pour bien comprendre le problème, Chandy et Lamport font une analogie avec la photographie. L'algorithme de prise d'instantané correspond à un groupe de photographes observant un panorama avec une scène dynamique comme par exemple le passage dans le ciel d'une compagnie d'oiseaux migrateurs. La scène est tellement vaste qu'elle ne peut être prise avec une seule photo. Les photographes doivent donc en prendre plusieurs et ensuite fusionner les différentes photos en une seule. Le problème est que les différentes photos ne peuvent être prises simultanément et que la scène que vous prenez en photo peut être modifiée pendant le processus. Par exemple, vous ne pouvez pas arrêter les oiseaux en plein vol le temps de prendre toutes vos photos ! Ainsi, il faut que la fusion de vos photos soit « cohérente » c'est-à-dire fidèle au panorama. Le problème est donc de définir formellement « cohérente » et ensuite déterminer comment prendre les différentes photos de manière à obtenir une fusion « cohérente ».

Propriétés stables. Pour valider leur approche, Chandy et Lamport proposent une vaste classe de propriétés détectables avec des prises d'instantané : soit $\mathcal{P}$ un prédicate (booléen) défini sur les configurations du système. $\mathcal{P}$ est dit *stable* si pour toute configuration γ, on a $\mathcal{P}(\gamma) \Rightarrow \mathcal{P}(\gamma')$ pour toute configuration γ' atteignable depuis γ.

2 Le modèle

Les auteurs considèrent des systèmes distribués où les communications ne sont pas nécessairement bidirectionnelles. En fait, un système distribué est décrit par un multigraphe dirigé étiqueté où les nœuds représentent les processus et les arcs représentent les canaux (unidirectionnel). Voir la figure 1 pour un exemple.

Les canaux sont supposés (pour simplifier) de capacités infinies et sans perte de message. De plus, ils sont FIFO : les messages sont délivrés dans l'ordre où ils sont envoyés. Cependant, ils sont asynchrones : le délai de transmission d'un message est fini mais quelconque.

La suite des messages reçus depuis un canal est donc un préfixe de la suite de messages envoyés dans ce canal. L'*état* d'un canal est la suite des messages envoyés dans le canal en excluant les messages reçus depuis ce canal.

Le comportement d'un processus est défini par un ensemble d'*états*, dont un *état initial*, et un ensemble d'*évènements*. Un évènement e sur un processus p est une action atomique qui peut changer l'état du processus p et l'état d'au plus un canal c incident à p : l'état de c peut être changé par l'envoi d'un message dans c (si c est un canal sortant de p) ou la réception d'un message depuis c (si c est un canal entrant de p). Un évènement e est défini par

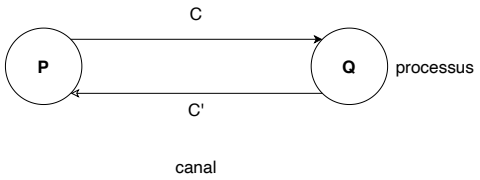

FIGURE 2 – Un système distribué avec deux processus et deux canaux.

1. le processus p chez qui l'évènement arrive,
2. l'état s de p immédiatement avant l'évènement,
3. l'état s' de p immédiatement après l'évènement,
4. le canal c (s'il y en a un) dont l'état est modifié par l'évènement, et
5. le message M (s'il y en a un) envoyé dans ou reçu depuis c.

Ainsi, e est un 5-uplet $\langle p, s, s', M, c\rangle$ où M et c peuvent être remplacés par la valeur $nulle$ si e ne modifie l'état d'aucun canal.

Une configuration d'un système distribué est un vecteur contenant l'état de chacun des processus et canaux du système. La configuration initiale du système est celle où chaque processus est dans son état initial et où tous les canaux sont vides. La survenue d'un évènement peut modifier la configuration du système. Soit $e = \langle p, s, s', M, c\rangle$ un évènement. On dit que e *peut survenir dans* la configuration γ si et seulement si

1. l'état de p dans γ est s et
2. si c est un canal entrant en p, alors l'état de c dans γ est une suite de messages avec M en tête.

On définit la fonction (partielle) $next$ où $next(\gamma, e)$ est la configuration du système immédiatement après la survenue de l'évènement e dans la configuration γ. La valeur de $next(\gamma, e)$ est définie seulement si l'évènement e peut survenir dans γ. Dans ce cas, $next(\gamma, e)$ est la configuration identique à γ sauf que

1. l'état du processus p dans $next(\gamma, e)$ est s',
2. si c est un canal entrant de p, alors l'état de c dans $next(\gamma, e)$ est l'état de c dans γ sauf que M a été supprimé de sa tête et
3. si c est un canal sortant de p, alors l'état de c dans $next(\gamma, e)$ est l'état de c dans γ où on a rajouté M à la fin.

Soit $seq = (e_i : 0 \leq i \leq k)$ une suite d'évènements. On dit que seq est une exécution (non-maximale) du système si et seulement si pour tout $i \in [0..k]$, e_i peut survenir dans la configuration γ_i, où γ_0 est la configuration initiale du système et, si $i > 0$ alors $\gamma_i = next(\gamma_{i-1}, e_{i-1})$.

Exemple 1. Pour illustrer la définition d'un système distribué, considérons un système simple avec deux processus **P** et **Q** et deux canaux **C** et **C'**, comme montré dans la figure 2. Dans ce système, un jeton (le message $Jeton$) circule en passant d'un processus à l'autre. Ce système est appelé « système à conservation de jeton unique ».

Chaque processus a deux états **S0** et **S1**, indiquant respectivement s'il ne détient pas ou s'il détient le jeton. L'état initial de **P** est **S1** et celui de **Q** est **S0** : initialement, **P** est l'unique détenteur d'un jeton. Chaque processus a deux évènements (*cf.* figure 3) :

1. $\langle$**P**,**S1**,**S0**,$Jeton$,**C**$\rangle$ pour **P** et $\langle$**Q**,**S1**,**S0**,$Jeton$,**C'**$\rangle$ pour **Q** : une transition de **S1** à **S0** avec un envoi de jeton dans l'unique canal sortant, et
2. $\langle$**P**,**S0**,**S1**,$Jeton$,**C'**$\rangle$ pour **P** et $\langle$**Q**,**S0**,**S1**,$Jeton$,**C**$\rangle$ pour **Q** : une transition de **S0** à **S1** avec réception de jeton depuis l'unique canal entrant.

Les configurations et les transitions possibles sont montrées dans la figure 4.

Une exécution du système correspond à un chemin dans le système de transition global (figure 4) à partir de la configuration initiale. Par exemple,

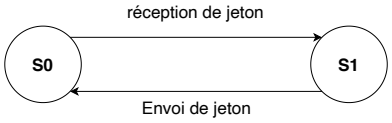

FIGURE 3 – Système de transition d'un processus.

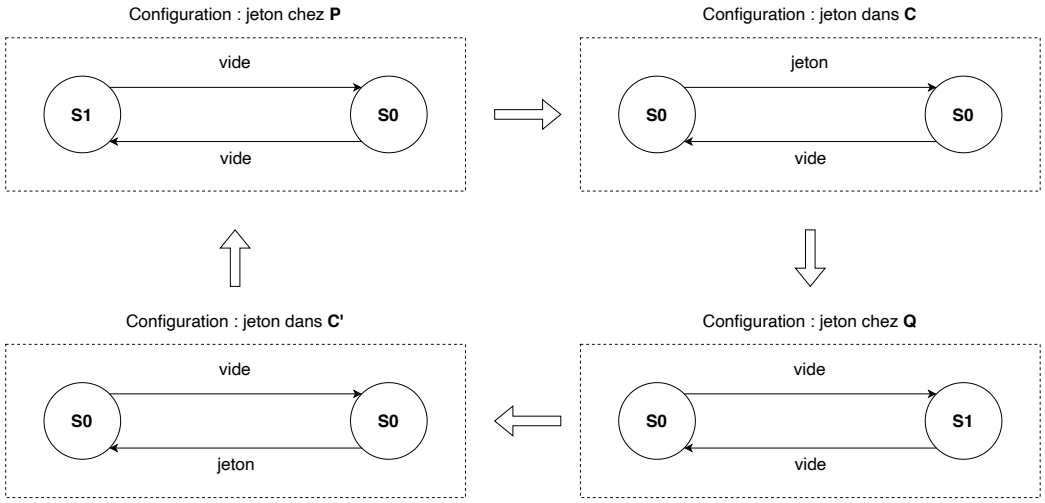

FIGURE 4 – Système de transition global du système à conservation de jeton unique.

1. la suite vide ou
2. la suite $\langle$**P**,**S1**,**S0**,$Jeton$,**C**$\rangle$, $\langle$**Q**,**S0**,**S1**,$Jeton$,**C**$\rangle$, $\langle$**Q**,**S1**,**S0**,$Jeton$,**C'**$\rangle$, c'est-à-dire, « **P** envoie le jeton, **Q** reçoit le jeton et **Q** envoie le jeton »

sont des exécutions (non-maximales) possibles du système. La suite suivante, quand à elle, n'est pas une exécution du système : $\langle$**P**,**S1**,**S0**,$Jeton$,**C**$\rangle$, $\langle$**Q**,**S1**,**S0**,$Jeton$,**C'**$\rangle$, c'est-à-dire, « **P** envoie le jeton, **Q** envoie le jeton ». En effet, l'évènement « **Q** envoie le jeton » ne peut pas survenir quand **Q** est dans l'état **S0**.

Dans la suite, les quatre configurations possibles seront nommées

1. dans-P,
2. dans-C,
3. dans-Q et
4. dans-C'

en fonction de la position du jeton dans le système.

3 L'algorithme

3.1 Justification des étapes de l'algorithme

L'algorithme de prise d'instantané fonctionne comme suit : chaque processus enregistre son propre état, et deux processus reliés par un canal collaborent pour enregistrer l'état du canal. Bien entendu, on ne peut pas assurer que les états de tous les processus et canaux seront enregistrés au même moment car le système est asynchrone et les processus n'ont pas accès à une horloge globale ; cependant l'algorithme assurera que les états enregistrés forment une configuration « cohérente » du système (qui sera caractérisée plus tard).

L'algorithme de prise d'instantané est superposé sur une exécution sous-jacente, c'est-à-dire, il doit être exécuté concurremment avec elle et ne doit pas la modifier.

Nous considérons maintenant un exemple qui justifie les étapes de l'algorithme. Dans cet exemple, nous allons supposer, de manière abusive, que nous pouvons enregistrer instantanément l'état d'un canal ; la discussion sur l'enregistrement d'un état de canal étant décalé à plus tard.

Soit **C** un canal du processus **P** au processus **Q**. Le but de cet exemple est d'obtenir une idée intuitive des relations entre l'instant auquel l'état de **C** est enregistré et les instants où les états de **P** et **Q** le sont.

Exemple 2. Considérons le système à conservation de jeton unique vu précédemment. Supposons que l'état de **P** enregistré par l'algorithme est son état dans-P. Alors, l'état enregistré de **P** indique qu'il détient le jeton. Supposons maintenant que la configuration change pour dans-C (parce que **P** a envoyé le jeton). Supposons que les états de **C**, **Q** et **C'** sont enregistrés alors que la configuration est dans-C. Ainsi, les états enregistrés indique que **C** contient un jeton alors que ni **C'** ni **Q** en possède un. Globalement, la configuration enregistrée contient deux jetons, un détenu par **P** et un en transit dans le canal **C**. Or, cette configuration n'est pas atteignable à partir de l'état initial du système à conservation de jeton unique !

Cette incohérence est due au fait que l'état de **P** a été enregistré avant l'envoi du jeton et que l'état de **C** a été enregistré après l'envoi de celui-ci. Soient $\#envoyé^P$ le nombre de messages envoyés dans **C** au moment où **P** enregistre son état et $\#envoyé^C$ le nombre de messages envoyés dans **C** au moment où l'état de **C** est enregistré. L'exemple suggère que la configuration enregistrée peut-être incohérente si $\#envoyé^P < \#envoyé^C$.

Considérons un autre scénario. Supposons que l'état de **C** est enregistré alors que la configuration est dans-P, qu'ensuite la configuration devienne dans-C et que les états de **C'**, **P** et **Q** sont enregistrés alors que la configuration est dans-C. La configuration enregistrée indique que le système ne contient aucun jeton. Cet exemple suggère que la configuration enregistrée peut être incohérente si l'état de **C** est enregistré avant que **P** envoie un message dans **C** et que l'état de **P** est enregistré après que **P** ait envoyé ce message dans **C**, c'est à dire, si $\#envoyé^P > \#envoyé^C$.

Ainsi à partir de ces deux exemples, nous apprenons qu'en général, une configuration enregistrée cohérente du système nécessite que

$$\#envoyé^P = \#envoyé^C. \tag{1}$$

Soient $\#recu^Q$ le nombre de messages reçus depuis **C** au moment où **Q** enregistre son état et $\#recu^C$ le nombre de messages reçus depuis **C** au moment où l'état de **C** est enregistré. On peut également déduire qu'une configuration enregistrée cohérente du système nécessite que

$$\#recu^Q = \#recu^C. \tag{2}$$

En effet, supposons que la configuration est dans-C lorsque l'état de **C** est enregistré. Supposons maintenant que la configuration est dans-Q (parce que **Q** a reçu le jeton) et que les états de **C'**, **P** et **Q** sont enregistrés. La configuration enregistrée indique que le système contient deux jetons. Cet exemple suggère que la configuration enregistrée peut être incohérente si l'état de **C** est enregistré avant que **Q** reçoive un message venant de **C** et que l'état de **Q** est enregistré après que **Q** ait reçu ce message depuis **C**, c'est à dire, si $\#recu^Q > \#recu^C$.

Considérons maintenant le scénario où la configuration est dans-C lorsque l'état de **Q** est enregistré. Supposons que la configuration devienne dans-Q (parce que **Q** a reçu le jeton) et que les états de **C'**, **P** et **C** soient enregistrés. La configuration enregistrée indique que le système ne contient aucun jeton. Cet exemple suggère que la configuration enregistrée peut être incohérente si l'état de **Q** est enregistré avant que **Q** reçoive un message venant de **C** et que l'état de **C** est enregistré après que **Q** ait reçu ce message depuis **C**, c'est à dire, si $\#recu^Q < \#recu^C$.

Dans toute configuration, le nombre de messages reçus depuis un canal ne peut excéder le nombre de messages envoyés dans ce canal, ainsi

$$\#envoyé^C \geq \#recu^C. \tag{3}$$

Des équations 1, 2 et 3, nous déduisons :

$$\#envoyé^P \geq \#recu^Q. \quad (4)$$

Ainsi, l'état d'un canal **C** qui doit être enregistré doit être la suite de messages envoyés au moment où l'état de l'émetteur est enregistré en excluant les messages reçus depuis ce canal avant que l'état du récepteur soit enregistré —c'est-à-dire, si $\#envoyé^C = \#recu^C$, alors l'état de **C** doit être la suite vide, et si $\#envoyé^C > \#recu^C$, alors l'état enregistré de **C** doit être la suite de messages à partir du $(\#recu^C + 1)^{\text{ème}}$ jusqu'au $(\#envoyé^C)^{\text{ème}}$ message envoyé par l'émetteur dans **C**. Ce fait et les équations 1-4 suggèrent un algorithme simple par lequel **Q** peut enregistrer l'état du canal **C**. Le processus **P** envoie un message spécial appelé *marqueur* dans **C** après le $(\#envoyé^P)^{\text{ème}}$ message qu'il a envoyé dans **C** (et avant d'envoyer d'autres messages dans **C**). Ce marqueur n'a aucun effet sur l'exécution sous-jacente. L'état de **C** a enregistré est alors la suite de messages reçus par **Q** après qu'il ait enregistré son propre état et avant de recevoir le marqueur envoyé dans **C**. Pour vérifier l'équation 4, **Q** doit enregistrer son état, s'il ne l'a pas déjà fait, après avoir reçu un marqueur depuis **C** et avant que **Q** ne reçoivent d'autres messages depuis **C** (sinon $\#recu^Q > envoyé^P$).

Cet exemple suggère les principes suivants pour réaliser l'algorithme de prise d'instantané.

3.2 Les grands principes de l'algorithme

La règle Marquage-Envoi pour un processus P. Pour chaque canal **C** entrant de **P** :

> **P** envoie *un* marqueur dans **C** après avoir enregistré son état et avant d'envoyer d'autres messages dans **C**.

La règle Marquage-Réception pour un processus Q. Sur réception d'un marqueur depuis le canal entrant **C** :

si Q n'a pas encore enregistré son état **alors**
  Q enregistre son état
  Q enregistre l'état de **C** comme la suite vide
sinon
  Q enregistre l'état de **C** comme la suite des messages reçus depuis **C**
    après que **Q** ait enregistré son état et
    avant que **Q** reçoive un marqueur depuis **C**.

3.3 Terminaison de l'algorithme

Les règles de marquage-envoi et de marquage-réception assurent que si un marqueur est reçu depuis chaque canal, alors chaque processus enregistrera son état et celui de ses canaux entrants (en particulier, les enregistrements non-atomiques de ses derniers termineront). Pour assurer que l'algorithme de prise d'instantané termine en temps fini, (`L1`) aucun marqueur ne doit rester pour toujours dans un canal entrant et (`L2`) chaque processus doit enregistrer son état en temps fini après l'initiation de l'algorithme (en effet, dans ce cas, on est sûr qu'au moins un marqueur est envoyé dans chaque lien, assurant ainsi avec `L1` qu'au moins un marqueur sera reçu depuis chaque canal). Puisque les canaux sont fiables (par hypothèse), `L1` est assuré puisque la réception des marqueurs est prévu par l'algorithme (règle de marquage-réception). Concentrons maintenant sur `L2`.

L'algorithme peut être initié par un ou plusieurs processus. Chacun d'entre-eux enregistre alors « spontanément » son état sans avoir reçu de marqueur au préalable. Si un processus **P** enregistre son état et qu'il existe un canal sortant de **P** à **Q**, alors **Q** enregistrera son état en temps fini car **P** enverra un marqueur dans le canal et ainsi **Q** recevra le marqueur en temps fini (`L1`). D'où, si **P** enregistre son état et qu'il existe un chemin entre **P** et un processus **R**, alors **R** enregistrera son état en temps fini car, par hypothèse de récurrence, chaque processus le long du chemin enregistrera son état en temps fini. Ainsi, la terminaison en temps fini est assurée si pour chaque processus **P** : soit **P** enregistre spontanément son état, soit il existe un chemin orienté depuis un processus qui enregistre spontanément son état jusqu'à **P**.

En particulier, si la topologie du système est *fortement connexe* et qu'il y a *au moins un initiateur*, alors tous les processus enregistrerons leur états en temps fini et donc l'algorithme terminera.

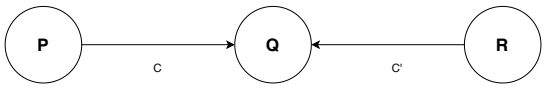

FIGURE 5 – Système où la terminaison n'est pas assurée.

Dans le cas où le réseau n'est pas fortement connexe, l'algorithme peut ne pas terminer : dans le système donné dans la figure 5, si **P** est l'unique initiateur, l'enregistrement de l'état du canal **C'** ne termine jamais.

Plus généralement, si le réseau n'est pas fortement connexe : soit l'algorithme termine mais oublie des processus, soit il ne termine pas. En fait, il ne termine pas s'il existe un processus inatteignable par un initiateur qui a pour successeur un processus qui lui est atteignable par un initiateur. Dans le cas mono-initiateur, il suffit d'avoir deux processus x et y tel qu'il existe un chemin P de x vers y mais pas de y vers x. Il existe alors un processus z dans le chemin P tel que z est atteignable depuis y mais pas son prédécesseur v dans P (*n.b.*, on peut avoir $z = y$). Dans ce cas, si y est l'unique initiateur, l'enregistrement du canal de v vers z par z ne terminera jamais.

L'algorithme décrit jusqu'à maintenant permet à chaque processus d'enregistrer son état et celui de ses canaux entrants. Ensuite, les états des processus et canaux doivent être collecté et rassemblé pour former une configuration. Un algorithme simple pour collecter des informations dans un système identifié où la topologie est fortement connexe est le suivant : chaque processus envoie l'information qu'il a enregistré taguée avec sa propre identité dans tous ses canaux sortants et chaque fois qu'il reçoit une information pour la première fois, il la copie et la relaie dans tous ses canaux sortants. Tous les informations enregistrée vont atteindre tous les processus en temps fini. Enfin, pour détecter la terminaison, les processus ont besoin de connaitre le nombre de processus n : après avoir collecté n enregistrements différents, un processus pourra déterminer une configuration du système.

Si maintenant on suppose que le réseau est bidirectionnel et il y a un unique initiateur, on peut simplement utiliser un PIR. L'initiateur peut démarrer le PIR après avoir exécuté spontanément sa règle de marquage-envoi. Puisque les canaux sont FIFO, un processus recevant un message du PIR depuis un canal c aura nécessairement reçu un marqueur *via* ce canal auparavant. Ainsi, les états des processus et des canaux peuvent être transmis à l'initiateur lors de la phase retour. Un processus enverra dans son message d'accusé réception son état, ceux de ces canaux entrants et les états transmis par ses fils.

4 Propriétés de la configuration calculée

Exemple 3. Pour comprendre intuitivement quelles sont les propriétés capturées dans la configuration calculée par l'algorithme de prise d'instantané nous allons considérer un système de deux processus **P** et **Q** reliés de manière bidirectionnelle et exécutant les systèmes de transitions donnés dans les figures 6 et 7. Supposons que l'état de **P** est enregistré alors que le système est dans la configuration **S0** (figure 8) ; donc l'état enregistré de **P** est **A**. Après avoir enregistré son état, **P** envoie son marqueur dans son unique canal sortant, disons c. Supposons maintenant que le système atteint la configuration **S1**, puis **S2** et enfin **S3** alors que le marqueur est encore en transit de **P** vers **Q**. Supposons que **Q** reçoive le marqueur alors que le système est dans **S3**, alors **Q** enregistre son état **D** et enregistre l'état de son canal entrant c' comme la suite vide. Ensuite, **Q** envoie un marqueur dans c'. À la réception du marqueur, **P** enregistre l'état de c' comme la suite contenant uniquement le message M'. La configuration enregistrée $\mathbf{S}^*$ est montrée dans la figure 9. L'algorithme de prise d'instantané ayant été initié depuis la configuration **S0** et ayant terminé dans la configuration **S3**.

Observons que la configuration $\mathbf{S}^*$ ne correspond à aucunes des configurations **S0**, **S1**, **S2**, **S3** qui sont apparues dans l'exécution. Ainsi, on peut légitimement se demander à quoi sert l'algorithme s'il peut enregistrer des configurations qui ne sont jamais apparues. Nous allons maintenant répondre à cette question.

Caractérisation de S^*. Soit $Seq = (e_i, i \geq 0)$ une exécution et soit S_i la configuration du système immédiatement avant la survenue de l'évènement e_i, avec $i \geq 0$, dans Seq. Supposons que l'algorithme de prise d'instantané est initié dans la configuration S_ℓ et qu'il termine dans la configuration S_ϕ, avec $0 \leq \ell \leq \phi$. Autrement dit l'algorithme est initié après $e_{\ell-1}$ (si $\ell > 0$) et avant e_ℓ ; de plus il termine après $e_{\phi-1}$ et avant e_ϕ. Nous avons observé dans l'exemple précédent que S^* peut être différente de toutes les configurations S_k avec $\ell \leq k \leq \phi$. Nous allons montrer maintenant

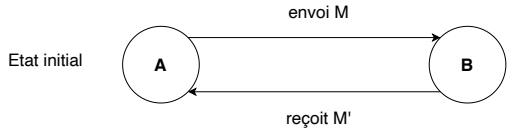

FIGURE 6 – Système de transition du processus **P**.

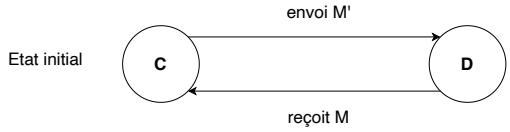

FIGURE 7 – Système de transition du processus **Q**.

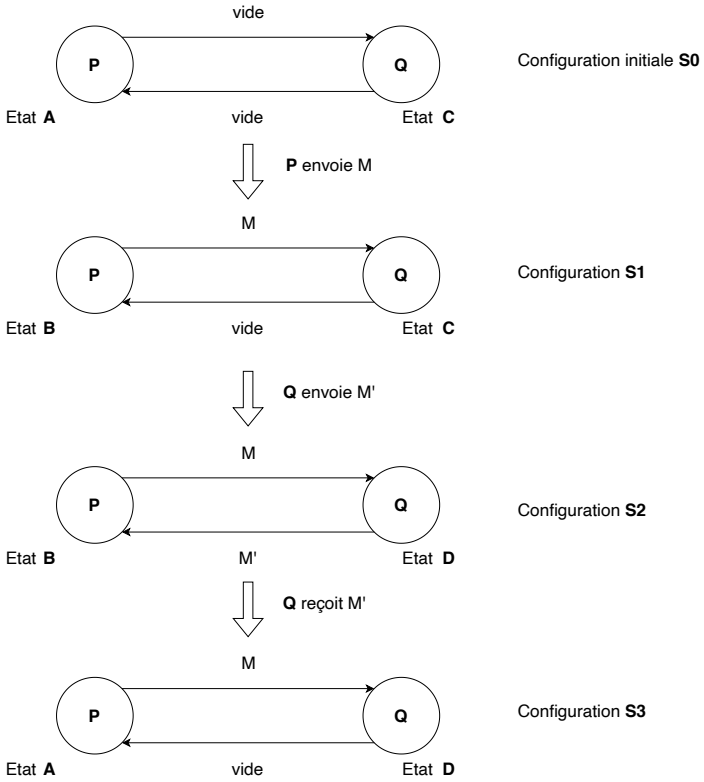

FIGURE 8 – Exécution possible du système.

FIGURE 9 – Configuration enregistrée par la prise d'instantané.

que :

1. S^* est atteignable depuis S_ℓ, et
2. S^ϕ est atteignable depuis S^*.

Précisément, il existe une exécution seq' où

1. seq' est une permutation de seq telle que S_ℓ, S^* et S_ϕ sont des configurations atteintes par seq',
2. $S_\ell = S^*$ ou S_ℓ apparaît avant S^*, et
3. $S_\phi = S^*$ ou S^* apparaît avant S_ϕ.

Théorème 1 *Il existe une exécution $seq' = (e'_i, i \geq 0)$ où*

1. *Pour tout i avec $i < \ell$ ou $i \geq \phi$: $e'_i = e_i$,*
2. *la sous-suite $(e'_i, \ell \leq i < \phi)$ est une permutation de la sous-suite $(e_i, \ell \leq i < \phi)$,*
3. *pour tout i avec $i \leq \ell$ ou $i \geq \phi$: $S'_i = S_i$, et*
4. *il existe k avec $\ell \leq k \leq \phi$ tel que $S^* = S'_k$.*

Preuve : Pour cette preuve, nous avons besoin de classer les évènements en deux types : les évènements *pré-enregistrement* et les évènements *post-enregistrement*. Soit e_i un évènement dans seq.

— e_i est appelé évènement *pré-enregistrement* si et seulement si le processus p où e_i est survenu a enregistré son état *après* e_i dans seq.

— e_i est appelé évènement *post-enregistrement* si et seulement si e_i n'est pas un évènement pré-enregistrement, c'est-à-dire, le processus p où e_i est survenu a enregistré son état *avant* e_i dans seq.

Remarquons que tous les évènements e_i avec $i < \ell$ sont des évènements pré-enregistrement et que tous les évènements e_i avec $i \geq \phi$ sont des évènements post-enregistrement.

Un évènement post-enregistré e_{j-1} peut être avant un évènement pré-enregistré e_j avec $\ell < j < \phi$. Dans ce cas, cela implique que e_{j-i} et e_j surviennent sur des processus différents (en effet, si e_{j-1} et e_j surviennent sur le même processus, alors le fait que e_{j-1} soit post-enregistré implique que e_j l'est aussi).

Nous allons maintenant construire une exécution possible seq' qui sera une permutation de seq où tous les évènements pré-enregistrés surviendront avant tous les évènements post-enregistrés. Nous montrerons aussi que S^* est une configuration atteinte dans seq' après que la survenue de tous les évènements pré-enregistrés et avant les évènements post-enregistrés.

Supposons qu'il y a un évènement post-enregistré e_{j-1} avant un évènement pré-enregistré e_j dans seq. Nous montrons maintenant que la suite obtenue en interchangeant e_{j-1} et e_j est une exécution possible. Tout d'abord, nous savons que e_{j-1} et e_j surviennent sur des processus différents. Soit p le processus où e_{j-1} survient et soit q le processus où e_j survient. L'évènement e_{j-1} ne peut contenir un envoi de message qui soit reçu lors de l'évènement e_j parce que, dans ce cas,

1. si un message a été envoyé dans un canal c quand e_{j-1} est survenu, alors un marqueur a été envoyé dans c *avant* e_{j-1} puisque e_{j-1} est un évènement post-enregistrement, et
2. si le message est reçu depuis le canal c quand e_j survient, alors le marqueur a été nécessairement reçu depuis c *avant* que e_j survienne (puisque les canaux sont FIFO), dans ce cas (par la règle de Marquage-Réception) e_j devrait être un évènement post-enregistrement aussi, une contradiction.

L'état du processus q n'est, par définition, pas modifié par la survenue de l'évènement e_{j-1} car ce dernier survient chez le processus p. Si e_j est un évènement où q reçoit un message M depuis le canal c, alors M devait être en tête de c avant l'évènement e_{j-1}, car un message envoyé lors de e_{j-1} ne peut pas être reçu lors de e_j (*cf.*, le point précédent). D'où, e_j peut survenir dans la configuration S_{j-1}.

Similairement, l'état du processus p n'est, par définition, pas modifié par la survenue de l'évènement e_j. Ainsi, e_{j-1} peut survenir après e_j. D'où, la suite d'évènements $e_0, \ldots, e_{j-2}, e_j, e_{j-1}$ est une exécution possible. De plus, la configuration après $e_0, \ldots, e_j$ est la même qu'après $e_0, \ldots, e_{j-2}, e_j, e_{j-1}$.

Soit seq^* la permutation de seq où seuls e_j et e_{j-1} ont été interchangé. D'après ce qui précède, seq^* est aussi une exécution possible. Soit $\bar{S}_i$ la configuration immédiatement après i évènements dans seq^* (attention les indices commencent à 0). D'après ce qui précède,

$$\bar{S}_i = S_i \text{ pour tout } i \neq j$$

En répétant l'échange des évènements post-enregistrés qui précèdent immédiatement des évènements pré-enregistrés, on finit par obtenir une permutation seq' de seq telle que :

1. Tous les évènements pré-enregistrement précèdent tous les évènements post-enregistrement,
2. seq' est une exécution possible,
3. pour tout i avec $i < \ell$ ou $i \geq \phi$: $e'_i = e_i$,
4. pour tout i avec $i \leq \ell$ ou $i \geq \phi$: $S'_i = S_i$.

Il nous reste à montrer que la configuration du système après tous les évènements pré-enregistrement et avant tous les autres dans seq' est S^*. Pour cela, nous devons montrer que :

1. l'état de chaque processus p dans S^* est le même que son état après la survenue de tous les évènements pré-enregistrement sur p, et
2. l'état de chaque canal c dans S^* est (la suite de messages correspondant aux envois pré-enregistrement dans c) $-$ (la suite de messages correspondant aux réceptions pré-enregistrement dans c).

La preuve du premier point est triviale, par définition (en effet, les évènements qui surviennent sur un processus donné sont dans le même ordre dans seq et dans seq'). Regardons maintenant le deuxième point. Soit c un canal de p vers q. L'état du canal c enregistré dans S^* est la suite des messages reçus par q *après* que q ait enregistré son état et *avant* que q reçoive un marqueur depuis c. La suite de messages envoyés par p dans c avant que p envoie un marqueur dans c est la suite correspondant aux envois pré-enregistrement dans c. Par suite on obtient le deuxième point. (Faire un dessin.)

Dans le détail, si l'état de q est enregistré après la réception du marqueur, tous les messages pré-enregistrement envoyés dans c ont été reçu par des réceptions pré-enregistrement (de q) dans c et donc (la suite de messages correspondant aux envois pré-enregistrement dans c) $-$ (la suite de messages correspondant aux réceptions pré-enregistrement dans c) donne la liste vide, ce qui correspond à l'état de c dans S^*.

Dans le cas contraire, la suite de messages correspondant aux envois pré-enregistrement dans c correspond aux messages envoyés par p dans c avant son marqueur. Seuls les messages de cette suite pourront être enregistré par q et ce, s'ils sont reçus par q après l'enregistrement de son état. Donc, état de c dans S^* exclura les éléments de la suite correspondant à des réceptions pré-enregistrement (de q) dans c. □

Exemple 4. Le but de cet exemple est de montrer comment une exécution seq' s'obtient à partir d'une exécution seq. Pour cela, nous revenons sur l'exemple 3. La suite d'évènements montrée dans l'exécution de la figure 8 est :

e_0 **:** **P** envoie M et change d'état pour B (un évènement post-enregistrement)

e_1 **:** **Q** envoie M' et change d'état pour D (un évènement pré-enregistrement)

e_2 **:** **P** reçoit M' et change d'état pour A (un évènement post-enregistrement)

Puisque e_0, un évènement post-enregistrement, précédent immédiatement e_1, un évènement pré-enregistrement, on peut les interchanger pour obtenir la permutation seq' suivante :

e'_0 **:** **Q** envoie M' et change d'état pour D (un évènement pré-enregistrement)

e'_1 **:** **P** envoie M et change d'état pour B (un évènement post-enregistrement)

e'_2 **:** **P** reçoit M' et change d'état pour A (un évènement post-enregistrement)

Dans seq', tous les évènements pré-enregistrement précèdent les évènements post-enregistrement. On peut remarquer que seq' est une exécution possible du système. De plus, S^* est la configuration juste après e'_0.

5 Détection de propriétés stables

Nous étudions maintenant comment résoudre la détection de propriétés stables en utilisant la prise d'instantané. Un algorithme de détection de propriétés stables peut être défini comme l'évaluation d'une sortie en fonction d'une entrée :

Entrée : Une propriété stable $\mathcal{P}$.

Sortie : Une valeur booléenne $Résultat$ vérifiant : $\mathcal{P}(S_\ell) \Rightarrow Résultat$ et $Résultat \Rightarrow \mathcal{P}(S_\phi)$

où S_ℓ et S_ϕ sont respectivement les configurations du système quand l'algorithme est initié et quand il termine.

Durant l'exécution, $\mathcal{P}(S)$ pour une configuration donnée S peut être déterminé par un processus qui doit appliquer le prédicat externe $\mathcal{P}$ sur une configuration collectée S. La sortie de l'algorithme étant la valeur booléenne $Résultat$, un processus désigné (par exemple un leader) doit prendre un état symbolisant le fait que la valeur de $Résultat$ est vraie et y rester, ou un autre état symbolisant que la valeur de $Résultat$ est fausse et y rester.

$Résultat = vrai$ implique que la propriété stable est vérifiée à la terminaison de l'algorithme. $Résultat = faux$ implique que la propriété stable n'est pas vérifiée à l'initiation de l'algorithme. En particulier, on ne peut pas déduire de $Résultat = faux$ que la propriété n'est pas vraie à la fin de l'algorithme.

Ainsi, une solution à la détection d'une propriété stable consiste à évaluer une configuration S^* à l'aide de la prise d'instantanée et de la stocker chez un processus désigné. Ce dernier devra évaluer $\mathcal{P}(S^*)$ et changer son état en conséquence pour refléter la valeur de $Résultat$.

La correction de l'algorithme de détection de propriétés stables est obtenue directement à partir des ces trois faits :

1. S^* est atteignable depuis S_ℓ (théorème 1),
2. S^ϕ est atteignable depuis S^* (théorème 1) et
3. $\mathcal{P}(\gamma) \Rightarrow \mathcal{P}(\gamma')$ pour toute configuration S' atteignable depuis S (par définition d'une propriété stable).

Références

[CL85] K. Mani CHANDY et Leslie LAMPORT. « Distributed Snapshots : Determining Global States of Distributed Systems ». *ACM Trans. Comput. Syst.*, 3(1) :63–75, 1985.