Abstraire à la volée les événements d'un système réparti

Thomas Gazagnaire — Claude Jard

IRISA/ENS Cachan-Bretagne,
Campus de Ker-Lann, F-35170 Bruz cedex, France
{Thomas.Gazagnaire, Claude.Jard }@bretagne.ens-cachan.fr

RÉSUMÉ. *Agréger les événements produits lors de l'exécution d'une application répartie est un problème ancien et trouve de nombreuses applications dans le débogage, la visualisation ou l'analyse des systèmes répartis. Nous reprenons ce problème en formalisant la notion d'équivalence préservant un ordre partiel. Ceci permet d'obtenir une notion d'abstraction hiérarchique qui peut être codée à la volée par un mécanisme d'horloge vectorielle.*

ABSTRACT. *Glueing events during the execution of a distributed application is an old problem, which can be used in many applications like debugging, visualisation or analysis of distributed systems. We revisit this problem in formalising the notion of equivalence preserving a partial ordering. This makes possible to define hierarchical abstractions, which can be coded on-the-fly by vector timestamps.*

MOTS-CLÉS : *Système réparti, abstraction, événement, horloge vectorielle, algorithme en ligne, ordre partiel*

KEYWORDS: *Distributed system, abstraction, event, vector timestamps, on-line algorithm, partial order*

1. Introduction

Une application répartie met en œuvre un ensemble de processus séquentiels autonomes coopérant dans un objectif commun. La coopération s'effectue par l'échange asynchrone de messages. Le comportement d'une telle application peut être modélisé par un ensemble d'événements. Un événement représente une action conduite par un processus. Il est supposé avoir lieu à un instant donné dans le référentiel du processus. Typiquement, les événements sont produits par une instrumentation du code : des ordres d'impression par exemple exécutés par des capteurs greffés sur les processus. Qualifions ces événements comme étant des événements de bas-niveau.

Le problème de l'abstraction consiste à agréger des événements de bas-niveau (ou "concrets") pour former des événements de plus haut niveau (ou "abstraits"). Il apparaît dans de nombreuses applications comme la supervision, le débogage, et plus généralement lorsqu'il s'agit de confronter le comportement observable d'un programme réparti à un modèle dans le cas où les événements de bas-niveau sont figés et où le modèle se situe à un niveau d'abstraction supérieur. La question est particulièrement pertinente lorsque l'agrégation s'effectue entre des événements produits sur des processus différents. Imaginons par exemple un modèle abstrait qui ne considère que la notion de message. Cette notion doit être reconstruite à partir de l'observation des événements d'émission et de réception. Grouper des événements pour créer des macro-événements cachant leur structure interne constitue cette notion d'abstraction d'événements, qui crée une représentation abstraite de l'exécution. Nous allons nous donner comme objectif que l'abstraction reste un ordre partiel et qu'elle puisse être codée à la volée par un observateur du système.

Il est bien connu depuis (Lamport, 1978) que l'ensemble des événements d'une exécution répartie s'organise sous la forme d'un ordre partiel de causalité, en considérant que deux événements sont ordonnés si ils ont lieu sur le même processus ou si ils sont séparés par un échange de messages. Deux événements non causalement ordonnés sont dits indépendants ("concurrent" en anglais). Une première proposition pour l'abstraction a été faite dans (Lamport, 1986). Elle propose qu'un événement abstrait précède un autre événement abstrait si et seulement si *tous* les événements concrets du premier événement abstrait précèdent *tous* les événements concrets du deuxième. Il est facile de montrer que l'ordre abstrait ainsi défini reste un ordre partiel. Le problème avec cette définition est que deux événements abstraits peuvent être indépendants alors qu'un événement concret du premier événement abstrait peut précéder causalement un événement concret du deuxième. Ce phénomène est contre-intuitif. Surtout, comme l'apparition d'une dépendance au niveau abstrait est exigeante, de nombreuses indépendances vont apparaître, détruisant la séquentialité des processus et rendant impossible la reconstruction de l'ordre concret à partir de l'ordre abstrait. La perte de cette séquentialité est problématique dans les applications de débogage, par exemple.

Les travaux successeurs (représentés par exemple par (Kunz, 1993)) prennent l'approche duale : un événement abstrait précède un autre événement abstrait si et seule-

ment *un* événement concret du premier précède *un* événement concret du deuxième. Cette idée est naturelle d'un point de vue mathématique puisqu'il s'agit de la notion classique de quotientage d'un graphe par une relation d'équivalence, les événements groupés formant les classes d'équivalence. Malheureusement, le quotient d'un ordre par une relation d'équivalence quelconque ne produit pas forcément un ordre (création de cycles par exemple). De notre point de vue il est dommageable que la structure de la causalité dans les exécutions réparties change de nature suivant le niveau d'abstraction considéré. En effet, cela rend impossible tout processus d'abstraction hiérarchique.

La solution adoptée dans (Ahuja *et al.,* 1990) consiste à imposer une restriction dans la façon de grouper les événements pour préserver la structure d'ordre partiel. Dans ces travaux, certaines classes d'équivalence sont considérées, qui sont définis comme les plus petits ensembles d'événements clos par communication (l'émission et la réception d'un message doivent être confinées dans l'atome) et sans trous (si deux événements produits par un processus donné sont dans l'atome, tous les événements intermédiaires produits sur ce processus le sont aussi). Il s'agit de la notion d'atome. Le prix à payer pour cette approche est que les atomes peuvent s'avérer très gros pour des exécutions complexes présentant de nombreux croisements intriqués de messages. L'autre inconvénient, symétrique de l'approche de Lamport, est que deux événements abstraits ordonnés peuvent chacun comprendre un événement concret indépendant de l'autre (par transitivité). Cet inconvénient semble mineur dans la mesure où la séquentialité des processus est préservée et où on pourrait reconstruire l'ordre concret.

Dans ce document, nous choisissons de généraliser cette dernière approche. Nous commençons par formaliser la notion d'abstraction préservant l'ordre partiel, notion paramétrée par un typage des événements. Puis, guidés par l'application à la supervision, nous étudions la question nouvelle du calcul au vol de l'abstraction pendant l'exécution des processus. Les algorithmes présentés dans cet article vont permettre d'utiliser le calcul d'une abstraction comme un filtre sur l'exécution et traiter des exécutions de longueur arbitraire. Chaque événement produit sur un processus est envoyé par un message vers un abstracteur unique. Charge à cet abstracteur de décider quand il a reçu suffisamment d'événements pour former un macro-événement. Il produit alors cet événement et la relation de causalité le reliant aux autres.

Le reste de l'article est organisé ainsi. Dans la partie 2, nous définissons la notion d'équivalence close par types et compatible avec l'ordre partiel (CTC-équivalence). La partie 3 présente le mécanisme d'extraction des événements et le codage de l'ordre par horloge vectorielle (filtrage). L'algorithme d'abstraction, localisé sur un processus observateur et produisant les macro-événements au vol est décrit dans la partie 4. La partie 5 discute les propriétés de ces algorithmes et leur originalité avant de conclure.

2. Définition formelle de l'abstraction

Considérons un ensemble de n processus séquentiels, que l'on note $\{P_i\}_{1 \leq i \leq n}$, qui communiquent à l'aide d'un ensemble de canaux fiables unidirectionnels. Chaque processus produit des événements atomiques, internes au processus (les événements

observables). Nous les indexons par un couple d'entier : le premier correspondant au numéro de processus sur lequel il a lieu, le second au nombre d'événements l'ayant précédé sur ce même processus. Nous notons ainsi $x_{i,j}$ le j^{eme} événement ayant lieu sur le processus P_i. L'ensemble des événements ayant lieu sur un processus P_i dans une exécution sera noté X_i, et l'ensemble de tous les événement noté $X = \bigcup_{1 \leq i \leq n} X_i$. On sait depuis (Lamport, 1978) que X peut être structuré par une relation d'ordre (partiel) entre les événements, notée $\leq$, qui capture la causalité entre les différents événements : $x_{i,j} \leq x_{k,l}$ signifie que $x_{i,j}$ s'est produit forcément avant $x_{k,l}$. Notons $\downarrow (x)$ les prédécesseurs causaux de l'événement x : $\downarrow (x) = \{y \in X, y \leq x\}$. On sait aussi depuis (Mattern, 1989, Fidge, 1991) que cette relation peut être codée au vol par une estampille vectorielle δ (un vecteur d'entiers), définie par $\delta(x) = (|\downarrow (x) \cap X_i\ |)_{i \in 1..n}$. $\delta(x)[i]$, la i^{eme} composante du vecteur $\delta(x)$, désigne le nombre d'événements qui précédent x sur P_i. Le plongement de l'ordre de causalité dans l'ordre sur les vecteurs d'entiers est attesté par la propriété suivante : $\forall x, y \in X, x \leq y \Leftrightarrow \forall i \in 1..n, \delta(x)[i] \leq \delta(y)[i]$. On notera par la suite, $max(\delta, \delta')$ le vecteur δ_{max} tel que $\delta_{max}[i] = max(\delta[i], \delta'[i])$ pour tout $i \leq n$. De plus, $\delta < \delta'$ si et seulement si $\delta[i] < \delta'[i]$ pour tout $i \leq n$. On rappelle la définition de la couverture : $\forall x, y \in X, x \prec y$ ssi $x \neq y \ \wedge \ \exists z \in X, x \leq z \leq y \Rightarrow (z = x \vee z = y)$. On dira que $\prec$ est la couverture de $\leq$.

Abstraire une structure consiste à fusionner des événements entre eux. Nous modélisons ceci par une relation d'équivalence $\sim \subseteq X \times X$. Deux événements seront fusionnés si ils sont équivalents. Ensuite, de manière classique, étant donnée l'équivalence $\sim$, nous notons $[x]_\sim$ la classe d'équivalence de x. La classe d'équivalence de x représente l'ensemble des événements qui seront fusionnés avec x lors de l'abstraction. Le résultat de l'abstraction est le quotient de X par $\sim$, noté $X/_\sim$, i.e. l'ensemble des classes d'équivalence. Un exemple est donné à la Figure 1a. Par la suite, nous identifierons une classe de $X/_\sim$ avec l'ensemble des événements qui la compose.

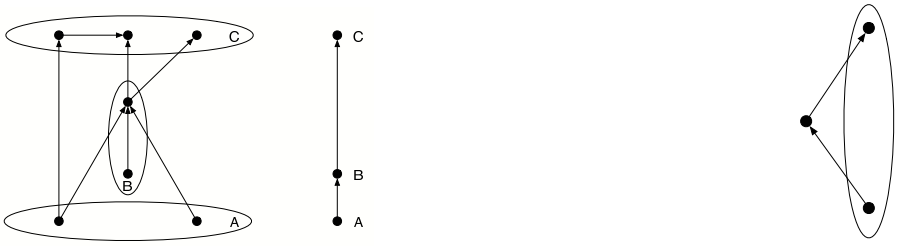


Figure 1. Figure a (gauche) : les événements entourés forment les macro-événements. Figure b (droite) : un quotient qui forme un cycle.

Définition 2.1 *Soit $(X, \prec)$ la couverture de la causalité d'une exécution distribuée. On note $\prec_\sim \subseteq X/_\sim \times X/_\sim$ la plus petite relation qui vérifie $x \prec y \implies [x]_\sim \prec_\sim [y]_\sim$. De plus, on notera $\leq_\sim$ la fermeture transitive de $\prec_\sim$.*

Malheureusement, le passage au quotient ne préserve par la structure de l'ordre sous-jacent. En effet, comme montré sur la Figure 1b, le quotient d'un ordre n'est

pas toujours antisymétrique, ni transitif. Notre objectif est de conserver une structure d'ordre sur l'objet abstrait. Pour cela, nous devons restreindre la relation $\sim$ pour qu'elle soit compatible avec le quotient.

Définition 2.2 *Une relation d'équivalence* $\sim \subseteq X \times X$ *est* compatible *avec la couverture de l'exécution* $(X, \leq)$ *(on dira plus simplement qu'elle est compatible avec l'exécution* $(X, \leq)$*) si elle satisfait la condition suivante : pour tout* $x_0 \dots x_{2n}$ *tels que* $x_{2i} \sim x_{2i+1}$ *et* $x_{2i+1} \prec x_{2(i+1)}$ *pour* $0 \leq i < n$*, si* $x_0 = x_{2n}$ *alors on a* $x_1 \sim \ldots \sim x_{2n}$.

On peut remarquer, de manière immédiate, que les deux abstractions triviales, correspondant à $\forall x \in X$, $[x]_\sim = \{x\}$ et $\forall x \in X$, $[x]_\sim = X$, sont compatibles avec n'importe quel ordre partiel.

Proposition 2.1 *Soit* $\sim$ *une relation d'équivalence compatible avec* $(X, \leq)$*. Alors* $\leq_\sim$ *est une relation d'ordre sur* $X/_\sim$.

La notion de compatibilité développée ici implique la notion d'abstraction convexe, définie dans (Basten *et al.,* 1997). Cependant, la notion d'abstraction convexe est trop fine pour que la structure d'ordre partiel soit préservée. En conséquence, ces travaux ne permettent pas de construire une abstraction qui soit un ordre partiel. Nous cherchons maintenant une famille d'abstractions compatibles non triviales. Nous proposons pour cela d'étiqueter les événements.

Définition 2.3 *Considérons un ensemble de types* T*, et une fonction de typage* $\mathcal{T}$ *étiquetant les événements de* X*. La relation d'équivalence* $\sim$ *est dite* close par types *si* $\forall x, y \in X$, $\mathcal{T}(x) = \mathcal{T}(y) \implies x \sim y$*. Une relation d'équivalence compatible avec* $(X, \leq)$ *et close par types sera appelée une* CTC-équivalence *pour* $(X, \leq)$.

Un premier exemple de typage est le suivant : on associe à chaque événement son numéro de processus : $\mathcal{T}(x_{i,j}) = i$. L'équivalence close par types correspondante construira le graphe de communication de l'exécution, c.a.d. le graphe orienté dont les sommets sont les processus, et il existe un arc entre deux processus si il existe une communication passant par le canal les interconnectant. Un graphe de communication peut-être cyclique, une relation close par types n'est donc pas a priori compatible. Un autre exemple de typage est de considérer une fonction qui donne un identifiant unique aux messages échangés. L'équivalence close par types correspondante consistera à fusionner les paires émission-réception d'un même message, et on obtiendra un graphe orienté dont les sommets correspondent aux messages.

Proposition 2.2 *L'ensemble des CTC-équivalences pour l'exécution* $(X, \leq)$ *est ordonné par inclusion en disant qu'une équivalence est plus fine qu'une autre si chaque classe d'équivalence de la première est incluse dans une classe d'équivalence de la seconde. L'ensemble des CTC-équivalences pour* $(X, \leq)$*, ordonné par la relation d'ordre définie ci-dessus, forme un treillis.*

L'élément minimal est appelé la *décomposition atomique* de l'exécution distribuée $(X, \leq)$. Dans le cas particulier où les événements de communication sont observés et où la fonction de typage associe un même type à l'envoi et à la réception d'un message, on retrouve la notion d'atome de (Ahuja *et al.,* 1990). Les classes de la CTC-équivalence $\sim_M$ correspondante, formant une partition de X, sont appelés des *atomes*. (Helouet *et al.,* 2000) a montré que l'on pouvait calculer la décomposition en atomes d'une exécution $(X, \leq)$ en temps $O(|X|^2)$, où $|X|$ est le nombre d'événements contenus dans X. Dans la partie suivante, nous généralisons cette approche à toute CTC-équivalence pour $(X, \leq)$, et nous proposons un algorithme centralisé qui permet d'abstraire une exécution à la volée, ce qui permet de traiter effectivement des exécutions de taille arbitraire.

3. Abstraction et horloges vectorielles

Dans cette partie, nous donnons une solution algorithmique au processus d'abstraction défini de manière formelle dans la partie précédente. Chaque processus est équipé d'un *capteur* qui implante le mécanisme d'horloges vectorielles en interceptant les messages émis et reçus. Lorsqu'un événement a lieu sur un processus, le capteur associé envoie un message à un *observateur* global. Le message est étiqueté par l'horloge vectorielle et le type de l'événement observé. Enfin, l'observateur global est en charge de produire à la volée, c'est à dire en même temps qu'il reçoit les messages provenant des différents capteurs, l'abstraction de l'exécution qui est en train de se dérouler sur les différents processus observés.

L'algorithme 1 définit le comportement du capteur attaché au processus P_i. Il reprend le calcul classique des horloges vectorielles en lui ajoutant une communication avec l'observateur global quand c'est nécessaire (ligne 4).

Algorithm 1 <Capteur> Gestion des événements liés au processus P_i

Input: n le nombre de processus, i le numéro du processus observé, $\mathcal{T}$ une fonction qui associe à chaque événement observable un type.

1: $\delta \leftarrow 0^n$;
2: **loop**
3: **if** un événement e observable se produit sur P_i **then**
4: Envoyer le message $(\mathcal{T}(e), \delta)$ à l'observateur global ;
5: $\delta[i] \leftarrow \delta[i] + 1$;
6: **end if**
7: **if** un message (m, δ_m) provenant du capteur associé à P_j est reçu **then**
8: Transmettre le message m à P_i ;
9: $\delta \leftarrow \max(\ \delta, \delta_m\ )$;
10: **end if**
11: **if** P_i veut envoyer un message m à P_j **then**
12: Envoyer le message (m, δ) au capteur associé à P_j ;
13: **end if**
14: **end loop**

Nous allons maintenant décrire le comportement de l'observateur global. Celui-ci reçoit donc des capteurs locaux les différents événements observés, sans garantie d'ordre, il peut y avoir des "trous" dans l'observation car certains messages ne sont pas encore arrivés du fait de l'asynchronisme des communications. Pour gérer cette contrainte, nous allons reprenons la notion d'horloges vectorielles pour les macro-événements introduite dans (Basten *et al.,* 1997). Ainsi, étant donné un ensemble A non vide d'événements concret que l'on regroupe en un unique macro-événement, nous lui associons deux horloges vectorielles $\Delta^-(A)$ et $\Delta^+(A)$, définies localement, c'est à dire qu'elles ne dépendent que des horloges vectorielles des événements contenus dans A.

Définition 3.1 $\Delta^-(A)[i] = min(\{+\infty\} \cup \{\delta(x_{i,j})[j] - 1 \mid x_{i,j} \in A\})$ *est le plus grand numéro d'apparition d'un événement précédant A sur l'instance P_i (et $+\infty$ s'il n'y a pas d'événement ayant lieu sur P_i).* $\Delta^+(A)[i] = max\{\delta(x)[i] \mid x \in A\}$ *est le plus grand numéro d'apparition d'un événement sur l'instance P_i vu par un événement de A.* $loc(A) = \{i \mid x_{i,j} \in A\}$ *est l'ensemble des processus impliqués dans l'exécution du macro-événement A.*

Nous nous intéressons maintenant à la relation de *précédence faible*, notée $\rightarrow$ définie de la manière suivante : $A \rightarrow B$ si et seulement si il existe $e \in A$, $e' \in B$ tels que $e \leq e'$. (Basten *et al.,* 1997) a montré que cette relation possède de bonnes propriétés algorithmique car on peut coder $\rightarrow$ ainsi que loc à l'aide des deux horloges Δ^- et Δ^+. En effet, $loc(A)$ peut être réécrit en l'ensemble $\{i \in \{1..n\} \mid \Delta^-(A)[i] < \Delta^+(A)[i]\}$ et $A \rightarrow B$ est équivalent à ce qu'il existe $i \in loc(A)$ tel que $\Delta^-(A)[i] < \Delta^+(B)[i]$. Nous noterons $\rightarrow^*$ la fermeture transitive de la relation $\rightarrow$. Soit $\mathcal{T}$ une fonction de typage sur les macro-événements définie comme suit : pour tout $A \subseteq X$, $\mathcal{T}(A)$ est l'ensemble $\{\mathcal{T}(x) \mid x \in A\}$. On note $A \rightarrow_{\mathcal{T}} B$ si et seulement $A \rightarrow B$ ou $\mathcal{T}(A) \cap \mathcal{T}(B) \neq \emptyset$. Enfin, nous noterons $\rightarrow^*_{\mathcal{T}}$ la fermeture transitive de $\rightarrow_{\mathcal{T}}$.

Etant donné une CTC-équivalence $\sim$ pour une exécution $(X, \leq)$, la proposition suivante montre que l'on peut utiliser les bonnes propriétés algorithmiques de $\rightarrow$ (que l'on peut coder en une comparaison de vecteurs d'horloge) pour calculer $\leq_\sim$.

Proposition 3.1 *Soit $\sim$ la CTC-équivalence minimale pour $(X, \leq)$. Alors $\rightarrow^*_{\mathcal{T}} = \leq_\sim$.*

Nous donnons ensuite un résultat important concernant l'abstraction d'exécutions partiellement observées.

Proposition 3.2 *Soient $\sim$ une CTC-équivalence compatible avec une exécution $(X, \leq)$. Pour toute observation partielle $Y \subseteq X$, et macro-événement A de $Y/_\sim$, A est dit* stable *pour l'observation Y si, de manière inductive :*

(i) *A est clos par types, c.a.d $\{x \in X \mid \exists x' \in A,\ \mathcal{T}(x) = \mathcal{T}(x')\} \subseteq A$,*

(ii) *A est convexe, c.a.d. $\{x \in X \mid \exists y, y' \in A,\ y \leq x \leq y'\} \subseteq A$,*

(iii) *tous les événements du passé causal de A sont dans des macro-événements qui sont stables.*

Alors, A stable pour Y est équivalent à A est un macro-événement de $X/_\sim$. On note $Stable(Y/_\sim)$ l'ensemble de tels A, et $Unstable(Y/_\sim)$ son complément dans $Y/_\sim$.

La partie stable d'une exécution le restera quoi qu'il se passe par la suite. Ainsi, dès qu'un macro-événement est détecté comme étant stable, il peut être produit par l'observateur. De plus, comme l'abstraction générée reste un ordre partiel, on peut coder ce macro-événement par une horloge vectorielle classique, en comptant cette fois-ci le nombre de macro-événements déjà générés sur chaque instance. On peut alors voir un macro-événement A comme un événement normal ayant lieu simultanément sur les instances de $loc(A)$. Dans la partie suivante, nous définissons un observateur qui produit à la volée des macro-événements et un mécanisme qui permet de les ordonner.

4. Algorithmes

Les événements observés par les capteurs sont reçus par l'observateur de manière complètement asynchrone, sans garantie de respect de l'ordre d'émission. Pour gérer cette difficulté, nous modélisons l'exécution globale par un ensemble X qui peut être infini et les événements observés par une famille d'ensemble d'événements $\emptyset = Y_0 \subset Y_1 \subset \ldots \subset Y_{|X|} = X$. Y_i correspond à l'ensemble des événements observés lorsque l'observateur global a reçu i messages provenant des capteurs. Les événements sont typés, et on cherche à construire à la volée $X/_\sim$, avec $\sim$ la CTC-équivalence minimale pour $(X, \leq)$.

L'algorithme 2 est la boucle principale de l'observateur. Il est appelé chaque fois qu'un nouveau message est reçu. Lorsque le k^{eme} message est reçu, l'observateur encapsule l'événement associé dans un macro-événement singleton A (ligne 12), puis il quotiente l'union de $Unstable(Y_{k-1}/_\sim)$ et de A (ligne 13). Finalement, il extrait les éléments stables de $\big(Unstable(Y_{k-1}/_\sim) \cup \{A\}\big)/_\sim$ afin de les fournir à l'utilisateur et il construit $Unstable(Y_k/_\sim)$ (ligne 14). Pour des raisons techniques, un macro-événement A sera codé comme la donnée de deux horloges $\Delta^-(A)$ et $\Delta^+(A)$, d'un entier $Size$ qui contiendra le nombre d'événements concrets agrégés dans A, et d'un multi-ensemble $\mathcal{T} = \{(t_i, n_i)\}_{i \leq \beta}$ où n_i est le nombre d'événements dans A ayant le type t_i et $\beta \leq k$ le nombre de types associés aux événements de $Unstable(Y_k/_\sim)$.

L'algorithme 3 insère le macro-événement singleton A dans $Unstable(Y_{k-1}/_\sim)$ et en calcule le quotient par $\sim$. La fusion de macro-événements (ligne 1 à 5) se fait en temps $O(n + \beta)$, les tests $A \rightarrow_{\mathcal{T}} B$ (lignes 7, 8 et 9) se font en $O(n + \beta)$ grâce au codage de la relation de précédence faible à l'aide des horloges Δ^- et Δ^+. Le choix d'une linéarisation de $\big(Unstable(Y_{k-1}/_\sim), \leq_\sim\big)$ (ligne 9) est un peu complexe car il faut reconstruire la fermeture transitive de $\rightarrow_{\mathcal{T}}$ (qui est égale à $\leq_\sim$ par la Proposition 3.1), ce qui peut-être fait en $O(n\alpha^2)$ en utilisant un algorithme de décomposition en niveau, où α est la taille maximale de $Unstable(Y_{i-1}/_\sim)$, pour $1 \leq i \leq k$ (et donc $\alpha \leq k$). Le calcul des horloges vectorielles (ligne 10) est classique : il consiste à calculer, pour chaque élément de L, le maximum de ces prédecesseurs immédiats par $\rightarrow_{\mathcal{T}}$, ce qui est fait en $O(n\alpha^2)$. Au total, l'algorithme INSERT à une complexité en $O(n\alpha^2 + \beta\alpha)$.

Algorithm 2 <Observateur> Programme principal

Input: i le numéro de processus qui sur lequel a lieu l'événement reçu, t son type et δ l'horloge vectorielle qui lui est associée.

Output: A est un macro-événement contenant l'unique événement considéré.

1: SINGLETON(i, t, δ) :=
2: $A.\Delta^+ \leftarrow \delta$;
3: $A.\Delta^- \leftarrow (+\infty)^n$;
4: $A.\Delta^-[i] \leftarrow A.\Delta^+[i] - 1$;
5: $A.Size \leftarrow 1$;
6: $A.\mathcal{T} \leftarrow \{(t, 1)\}$;
7: **return** A ;

Input: n le nombre de processus, $\mathcal{B}$ une fonction qui à chaque type associe le nombre d'événements étiquetés par ce type.

8: MAIN(n, $\mathcal{B}$) :=
9: $Memory \leftarrow \emptyset$; $MaxStableEvent \leftarrow 0^n$; $OutClock \leftarrow 0^{n \times n}$;
10: **loop**
11: **if** le message (t, δ) numéro k provenant du capteur associé à P_i est reçu **then**
12: $A \leftarrow$ SINGLETON(i, t, δ) ;
/* $Memory = Unstable(Y_{k-1}/_\sim)$ */
13: $Memory \leftarrow$ INSERT(A, $Memory$) ;
/* $Memory = \big(Unstable(Y_{k-1}/_\sim) \cup \{A\}\big)/_\sim$*/
14: $(Memory, MaxStableEvent, OutClock) \leftarrow$
EXTRACT($Memory$, $MaxStableEvent$, $OutClock$, $\mathcal{B}$) ;
/* $Memory = Unstable(Y_k/_\sim)$ */
15: **end if**
16: **end loop**

L'algorithme 4 extrait de $\big(Unstable(Y_{k-1}/_\sim) \cup \{A\}\big)/_\sim$ les macro-événements stables et construit $Unstable(Y_k/_\sim)$. Les tests effectués ligne 7 correspondent aux points (i), (ii) et (iii) de la Proposition 3.2. La condition (i) nous impose de rajouter une contrainte supplémentaire sur le type d'abstraction considéré. En effet, pour savoir si un macro-événement est clos par types, il faut avoir des informations sur le type de tous les événements qui pourront arriver dans le futur, ce qui est impossible dans le cas général. Nous supposons donc ici que l'on connaît une fonction $\mathcal{B}$ qui associe à chaque type le nombre d'événements ayant ce type. Cette supposition n'est pas si restrictive, elle permet par exemple d'exprimer la notion d'atome définie dans (Ahuja *et al.,* 1990), avec $\mathcal{B}(t) = 2$ si t est associé à l'envoi et à la réception d'un même message, 1 sinon. La complexité de CLOSE est en $O(\beta)$, celle de la ligne 9 en $O(n^2)$. La complexité de EXTRACT est donc en $O(\alpha n^2 + \alpha\beta)$.

Proposition 4.1 *Soient n le nombre de processus observés, $\alpha = |Unstable(Y/_k/_\sim)|$ et β le nombre de types associés aux événements de $Unstable(Y/_k/_\sim)$. Alors, l'al-*

Algorithm 3 <Observateur> INSERT(A, $Memory$)

Input: A et B sont deux macro-événements.
Output: A est modifié et contient l'union de A et B.

FUSION(A, B) :=
  $A.\Delta^- \leftarrow$ min($A.\Delta^-$, $B.\Delta^-$) ;
  $A.\Delta^+ \leftarrow$ max($A.\Delta^+$, $B.\Delta^+$) ;
  $A.Size \leftarrow A.Size + B.Size$;
  $A.\mathcal{T} \leftarrow A.\mathcal{T} \cup B.\mathcal{T}$;

Input: A est le nouveau singleton à ajouter à $Memory = Unstable(Y_{k-1}/_\sim)$.
Output: $\big(Unstable(Y_{k-1}/_\sim) \cup \{A\}\big)/_\sim$.

INSERT(A, $Memory$) :=
  $ImmPred \leftarrow \{B \mid A \rightarrow_{\mathcal{T}} B\}$;
  $ImmSucc \leftarrow \{B \mid B \rightarrow_{\mathcal{T}} A\}$;
  $L \leftarrow$ Choisir une linéarisation de $(Memory, \rightarrow^*_{\mathcal{T}})$;
  Parcourir L pour calculer une horloge vectorielle $\delta(B)$ pour chaque $B \in L$;
  $Pred \leftarrow \{B \mid \exists C \in ImmPred,\ \delta(B) \leq \delta(C)\}$;
  $Succ \leftarrow \{B \mid \exists C \in ImmSucc,\ \delta(C) \leq \delta(B)\}$;
  for all $B \in Pred \cap Succ$ **do** FUSION(A,B) **end do** ;
  return $\big(M - (Pred \cap Succ)\big) \cup \{A\}$;

gorithme 2, qui passe de $Unstable(Y/_{k-1}/_\sim)$ à $Unstable(Y_k/_\sim)$ en produisant les macro-événements stables correspondants, a une complexité en $O(\alpha(n\alpha + n^2 + \beta))$.

En pratique, le désordre de réception des messages sera réduit, ou en tout cas borné. Ce qui implique que α et β sont bornés eux aussi. Dans ce cas, la complexité de l'algorithme d'abstraction présenté ci-dessus est indépendant de la taille de l'exécution à abstraire et peut donc être réellement utilisé.

5. Conclusion

Notre contribution est à situer principalement en continuité des travaux de (Basten *et al.,* 1997) et (Helouet *et al.,* 2000). (Basten *et al.,* 1997) a introduit la notion de précédence faible et son codage vectoriel. (Helouet *et al.,* 2000) a défini la notion de décomposition atomique dans les "Message Sequence Charts" et proposé un algorithme hors-ligne fondé sur le calcul de composantes fortement connexes d'un graphe. Nous améliorons ces travaux sur les points suivants. D'une part, nos algorithmes marchent à la volée sur un observateur implanté dans un processus du système. Gérer les événements qui arrivent dans le mauvais ordre demande un soin particulier (notion de macro-événement en cours de formation et détection de la stabilité). D'autre part, la construction de nos macro-événements est paramétrée par une fonction de typage, et par l'obligation que nous nous donnons d'avoir un ordre partiel clos par types. Cela présente une généralisation intéressante aussi bien au niveau théorique que pratique.

Algorithm 4 <Observateur> EXTRACT($Memory$, $MaxStableEvent$, $OutClock$, $\mathcal{B}$)

CLOSE(B, $\mathcal{B}$) := **for all** $t^i \in B.\mathcal{T},\ \mathcal{B}(t) = i$;
CONVEX(B) := $\sum_{i \in loc(B)} (B.\Delta^+[i] - B.\Delta^-[i]) = Size(B)$;
MINIMAL(B, $MaxStablevent$) :=
  for all $i \in loc(B),\ B.\Delta^-[i] = MaxStableEvent[i]$;

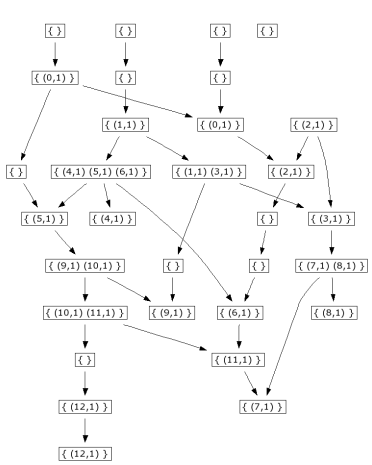
Input: Un ensemble de macro-événements $Memory = \big(Unstable(Y_{k-1}/_\sim) \cup \{A\}\big)/_\sim$, un vecteur d'entiers $MaxStableEvent$ qui contient pour chaque processus le numéro de l'événement maximal ayant déjà été extrait après avoir reçu $k-1$ messages, une matrice d'entiers $OutClock$ qui contient, pour chaque processus le nombre de macro-événements stables ayant été vus sur chaque processus après avoir reçu $k-1$ messages ($OutClock[i][j]$ correspond au nombre de macro-événements que P_i a vu sur P_j), une fonction $\mathcal{B}$ qui associe à chaque type son nombre d'occurrence.

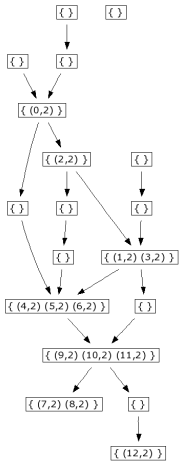
Output: L'ensemble $Unstable(Y_k/_\sim)$, un vecteur d'entiers $MaxStableEvent$ qui contient pour chaque processus le numéro de l'événement maximal qui a déjà été extrait après avoir reçu k messages, une matrice d'entiers $OutClock$ qui contient, pour chaque processus, le nombre de macro-événements stables ayant été vus sur chaque processus après avoir reçu k messages.

EXTRACT($Memory$, $MaxStableEvent$, $OutClock$, $\mathcal{B}$) :=
  $\delta \leftarrow 0^n$;
  while exists B in $Memory$ **such that**
      CLOSE(B, $\mathcal{B}$) & CONVEX(B) & MINIMAL(B, $MaxStableEvent$) **do**
    for all $i \in loc(B)$ **do** $MaxStableEvent[i] \leftarrow B.\Delta^+[i]$ + 1 **end do** ;
    $\delta \leftarrow$ max $\{OutClock[i] \mid i \in loc(B)\}$;
    for all $i \in loc(B)$ **do** $\delta[i] \leftarrow \delta[i] + 1$ **end do** ;
    for all $i \in loc(B)$ **do** $OutClock[i] \leftarrow \delta$;
    $Memory \leftarrow Memory - \{B\}$;
    Produire le macro-événement stable (B, δ) ;
  end do
  return ($Memory$, $MaxStableEvent$, $OutClock$) ;

Finalement le fait de rester strictement dans le cas d'ordres partiels permet de reproduire le schéma d'abstraction à plusieurs niveaux, offrant ainsi une fonction d'abstraction hiérarchique. L'observateur agit comme un filtre sur le système produisant les événements de plus bas niveau. On peut donc rebrancher un observateur sur un observateur et ainsi de suite.

Les différents algorithmes ont été mis en œuvre. La figure suivante montre un exemple d'abstraction produite à partir d'une exécution aléatoire. Les événements (à gauche) et macro-événements (à droite) sont représentés par des rectangles et sont étiquetés par leur type.





Nous pensons que les complexités obtenues sont raisonnables dans le cas pratique où le désordre devant l'observateur reste petit. La mise en œuvre de ces algorithmes dans un vrai système rend possible l'abstraction en ligne et hiérarchique des événements, besoin que nous pensons intéressant. Dans cette perspective, nous envisageons de distribuer les algorithmes de l'observateur sur chacun des processus du système.

6. Bibliographie

Ahuja M., Kshemkalyani A. D., Carlson T., « A Basic Unit of Computation in Distributed Systems. », *ICDCS*, p. 12-19, 1990.

Basten T., Kunz T., Black J. P., Coffin M. H., Taylor D. J., « Vector time and causality among abstract events in distributed computations », *Distrib. Comput.*, vol. 11, n° 1, p. 21-39, 1997.

Fidge C., « Logical Time in Distributed Computing Systems », *Computer*, vol. 24, n° 8, p. 28-33, 1991.

Helouet L., Le Maigat P., « Decomposition of Message Sequence Charts », *in* , S. Graf, , C. Jard (eds), *2nd Workshop on SDL and MSC (SAM2000)*, Grenoble, France, June, 2000.

Kunz T., « Issues in Event Abstraction », *PARLE '93 : Proceedings of the 5th International PARLE Conference on Parallel Architectures and Languages Europe*, Springer-Verlag, London, UK, p. 668-671, 1993.

Lamport L., « Time, Clocks, and the Ordering of events in a Distributed System », *Communications of the ACM*, vol. 21, n° 7, p. 558-565, July, 1978.

Lamport L., « On Interprocess Communication », *Distributed Computing*, vol. 1, p. 77-101, 1986.

Mattern F., « On the relativistic structure of logical time in distributed systems », *Parallel and Distributed Algorithms*p. 215-226, 1989.