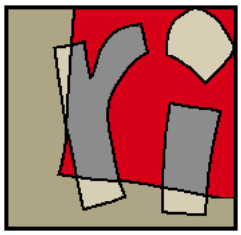

UNIVERSITÉ DE PARIS-SUD

ÉCOLE DOCTORALE 427 :
INFORMATIQUE PARIS-SUD

LABORATOIRE : LRI

THÈSE

INFORMATIQUE

PAR

Peva BLANCHARD

Synchronization and Fault-tolerance in Distributed Algorithms

Date de soutenance : 24/09/2014

Composition du jury :

Directeur de thèse :	M. Joffroy BEAUQUIER	Professeur (LRI, Paris XI)
Co-encadrante :	Mme. Sylvie DELAËT	Maître de Conférences (LRI, Paris XI)
Président du jury :	Christine PAULIN	Professeure (LRI, Paris XI)
Rapporteurs :	Rachid GUERRAOUI	Professeur (LPD, EPFL)
	Luis RODRIGUES	Professeur (DEI, Univ. de Lisboa)
Examinateurs :	Hugues FAUCONNIER	Maître de Conférences (LIAFA, Paris VII)
Invités :	Janna BURMAN	Maître de Conférences (LRI, Paris XI)

Mots-clés. *algoritmique répartie, protocoles de population, tolérance aux défaillances, auto-stabilisation, collecte de données, consensus, élection de leader, réplication, oracles.*

Résumé Dans la première partie de ce mémoire, nous étudions le modèle des protocoles de population, introduit dans [4]. Ce modèle permet de représenter les grands réseaux de capteurs (ou agents) mobiles anonymes dotés de faibles ressources. Les contraintes de ce modèle sont si sévères que la plupart des problèmes classiques d'algoritmique répartie, tels que la collecte de données, le consensus ou l'élection d'un leader, sont difficiles à analyser, sinon impossibles à résoudre.

Nous commençons notre étude par le problème de collecte de données. Celui-ci consiste principalement à transférer des valeurs réparties dans la population d'agents mobiles vers une station de base en un minimum de temps (temps de convergence). En utilisant une hypothèse d'équité, dite hypothèse de temps de couverture et introduite dans [4], nous calculons des bornes optimales sur le temps de convergences de différents protocoles concrets. Ensuite, nous étudions le problème du consensus et d'élection de leader. Il a été montré que ces problèmes sont impossibles à résoudre dans le modèle original des protocoles de population. Pour contourner cette impossibilité, il est possible d'adjoindre au modèle certaines hypothèses sous la forme d'oracles. Nous proposons ensuite divers oracles permettant de résoudre le problème du consensus et d'élection de leader dans divers environnements, et nous étudions leurs puissances relatives. Ce faisant, nous développons un cadre formel permettant de représenter toutes les variétés d'oracles introduites, ainsi que leur possibles relations.

Dans la seconde partie de ce mémoire, nous étudions le problème de la réplication de machine à états finis dans le modèle (classique) de communications asynchrones à passage de message. L'algoritme Paxos, introduit dans [12,13] est une solution (partielle) bien connue au problème de la réplication capable de tolérer des pannes crash. Notre contribution, dans cette partie, consiste à améliorer Paxos afin qu'il puisse également tolérer des défaillances transitoires. Ce faisant, nous définissons la notion de machine répliquée pratiquement autostable.

1 Introduction

Les technologies de communication, depuis les réseaux de capteurs mobiles jusqu'aux bases de données répliquées à l'échelle mondiale, font l'objet depuis plusieurs années d'une attention considérable. Une des tâches principales qui incombent à la communauté de l'algoritmique répartie consiste à développer des méthodes théoriques suffisamment pointues pour exploiter ces nouveaux types de réseaux.

Un lieu fondamental de la réflexion dans notre domaine est l'opposition entre *synchronisme* et *asynchronisme*. D'un point de vue très général, un système réparti, un réseau, se conçoit comme un ensemble de processeurs séquentiels équippés de divers moyens de communication. Ces moyens incluent, par exemple, le

passage de message, la mémoire partagée, etc. La notion classique de synchronisme et d'asynchronisme se réfère à la plus ou moins grande liberté dans l'entrelacement des transitions locales des processeurs. Dans un système purement synchrone, une horloge globale imprime aux processeurs une cadence fixée, c'est-à-dire, ils exécutent leurs pas de calcul en même temps. À l'opposé, dans un système asynchrone, chaque processeur exécute ses pas de calcul indépendamment de la marche des autres processeurs ; un processeur peut mettre un temps arbitrairement long à exécuter telle transition élémentaire. Cette définition concrète est un cas particulier d'une situation plus générale. L'*asynchronisme* peut être vu comme une certaine indépendance relative des transitions locales des processeurs, tandis que le *synchronisme* peut se voir, à l'opposé, comme une relative dépendance des processeurs entre eux. De manière qualitative, on peut associer à un réseau donné, un certain niveau de synchronisme correspondant à une certaine mesure de la relative dépendance des processeurs entre eux. Bien sûr, cette définition qualitative comprend la définition usuelle de synchronisme et d'asynchronisme (exprimés en termes de délais, ou de périodicité), mais elle s'applique à des cas plus généraux.

Un point important de notre domaine tient au fait qu'un problème d'algoritmique répartie consiste généralement en la définition d'algoritmes *locaux* imposant aux processeurs de satisfaire des spécifications *globales*. Cependant, intuitivement, si les processeurs sont trop indépendants les uns des autres, il y a peu de chance qu'ils puissent se coordonner pour accomplir la tâche globale considérée. Autrement dit, si le niveau de synchronisme est trop faible, alors le problème visé peut être impossible à résoudre à l'aide d'algoritmes locaux. Par contre si le niveau de synchronisme est suffisant, une solution peut exister ; et si le niveau de synchronisme est très élevé, une solution plus efficace encore peut exister.

Il y a essentiellement deux façons d'estimer le niveau de synchronisme d'un réseau. On peut d'abord s'appuyer sur des hypothèses *explicites*, e.g., délais de communication bornés, (dans le cas d'un réseau mobile) mouvement des agents connus, types de défaillance considérés, etc. Dans ce cas, l'approche est "ascendante" : on fixe un certain niveau de synchronisme, et on essaie de définir un algoritme permettant de résoudre le problème considéré. Si le problème s'avère être impossible à résoudre, on peut viser une version affaiblie du problème.

Une autre approche consiste à poser des hypothèses *implicites*. On ajoute au réseau un oracle, une sorte de boîte noire qui rend un certain service dont on ne connait que la spécification globale ; on ne connait pas et on ne cherche pas à savoir comment ce service est implémenté. Cet oracle va, d'une certaine manière, augmenter le niveau de synchronisme du réseau sans se référer aux conditions explicites de son implantation. L'approche est alors "descendante" : on fixe le problème, et on cherche le niveau de synchronisme minimal (sous forme d'oracle) permettant de résoudre ce problème.

De nombreux résultats fondamentaux d'algoritmique répartie illustrent les remarques précédentes. Le problème du consensus s'inscrit parfaitement dans cette discussion : intuitivement, faire en sorte que les processeurs s'accordent

sur une même valeur requiert un niveau de synchronisme relativement élevé. Le fameux article de Fischer, Lynch et Paterson [11] montrent qu'un réseau asynchrone à passage de message susceptible de pannes définitives (crash) ne présente pas un niveau de synchronisme suffisant : une seule panne crash peut compromettre le système entier. Plus tard, Dwork, Lynch et Stockmeyer [9] énumèrent des hypothèses explicites permettant de résoudre le problème du consensus dans ce type de réseaux, illustrant ainsi l'approche explicite mentionnée ci-dessus. L'approche implicite est inaugurée par l'article [6]. Les auteurs introduisent la notion de détecteur de défaillances, c'est-à-dire, d'oracles renseignant les processeurs, avec plus ou moins de fiabilité, au sujet des pannes qui se sont produites. En ajoutant de tels oracles au réseau, ils fournissent un algoritme qui résoud le consensus. Cependant, leur résultat le plus important tient au fait qu'ils ont exhibé l'oracle minimal permettant de résoudre le consensus. Et bien que la notion d'oracle minimal soit problématique [7], c'est la première occurrence de l'approche implicite en algoritmique répartie ; du moins, autant que nous sachions.

Ce mémoire s'inspire des approches précédentes (tant explicites qu'implicites) dans l'étude de deux modèles différents. Le premier modèle, les protocoles de population [2], permet de représenter des réseaux de capteurs mobiles dotés de faibles ressources. Nous y étudions les problèmes de collecte de données, de consensus et d'élection de leader autostabilisante. Le deuxième est le modèle classique de réseau asynchrone à passage de message avec possibilité de pannes définitives. Nous y étudions le problème de la réplication autostabilisante de machine finie. Nous donnons dans la suite un résumé de nos contributions principales.

2 Protocoles de population

2.1 Modèle

(Définition). Les protocoles de population [2] sont un modèle de grands réseaux composés d'agents mobiles anonymes et dotés d'une faible mémoire. On peut se représenter les choses en imaginant une population d'agents qui se déplacent plus ou moins aléatoirement dans un certain espace géographique, de sorte que deux d'entre eux ne peuvent communiquer que lorsqu'ils sont suffisamment proches. Le protocole proprement dit consiste en une liste finie de règles de transition décrivant comment les états de deux agents sont mis à jours lorsqu'ils se rencontrent. On représente les possibilités de rencontre entre agents à l'aide d'un graphe, appelé graphe de communication : chaque noeud y représente un agent, et un arc indique la possibilité de rencontre entre deux agents. La mobilité des agents est alors représenté par un ensemble de conditions sur l'ordonnancement des rencontres entre agents ; ces conditions forment l'hypothèse d'équité. Par exemple, on peut considérer qu'un ordonnancement équitable est une séquence de rencontres telle que deux agents quelconques liés dans le graphe de communication se rencontrent infiniment souvent. Un dernier aspect important vient du fait que les agents ont une mémoire de taille indépendante de la taille de la population, et qu'ils ne connaissent pas le graphe de communication sous-jacent.

De manière plus formelle, un protocole de population $\mathcal{A}$ comprend un espace d'états $States(\mathcal{A})$, un alphabet d'entrée $In(\mathcal{A})$, un alphabet de sortie $Out(\mathcal{A})$, et un ensemble (fini) de règles de la forme $q_1, q_2 \xrightarrow[o_1,o_2]{i_1,i_2} q_1', q_2'$. Intuitivement, cette règle signifie que si deux agents dans les états q_1, q_2 se rencontrent et que des valeurs d'entrée i_1, i_2 (e.g. données par un capteur) leurs sont fournies, alors ils passent dans les états q_1', q_2' et produisent des valeurs de sortie o_1, o_2 respectivement.

Étant donné un graphe de communication G, une exécution du protocole $\mathcal{A}$ sur G est une séquence

$$\gamma_1 \xrightarrow[out_1]{e_1,in_1} \gamma_2 \xrightarrow[out_2]{e_2,in_2} \dots$$

où les γ_* sont des configurations (assignation d'un état à chaque agent de la population), les e_i sont des arcs de G interprétés comme des rencontres entre agents, les in_* et out_* sont des couples de valeurs d'entrée et de sortie respectivement. La transition entre deux configurations successives s'effectue en modifiant les deux agents concernés par la règle correspondante du protocole.

La séquence de rencontres $e_1 e_2 \dots$ forme *l'ordonnancement* sous-jacent de l'exécution. Cette même séquence de rencontres annotée avec les valeurs d'entrée, $(e_1, in_1)(e_2, in_2) \dots$ forme *l'historique d'entrée* de l'exécution. On définit de la même façon *l'historique de sortie* de l'exécution.

(Oracles). Comme expliqué ci-dessous, les conditions du modèle des protocoles de population sont si sévères que certains problèmes classiques d'algoritmique répartie (consensus, élection de leader) sont impossibles à résoudre dans bon nombre de situations. Pour contourner ces résultats, il est possible, à la manière des détecteurs de défaillances de Chandra et Toueg [6], d'introduire la notion d'oracle.

Un oracle est une sorte de boîte noire qu'on ajoute au système, et qui permet à ses composants d'obtenir certaines informations globales à propos du système. Par exemple, les détecteurs de défaillance de Chandra et Toueg renseignent les processeurs du système, avec plus ou moins de fiabilité, au sujet des arrêts inopinés de certains des processeurs. Autrement dit, un oracle se définit par une certaine spécification globale, un certain ensemble d'hypothèses reliant les événements du système et les sorties produites par l'oracle.

Cependant, pour éviter les solutions triviales, comme supposer un oracle qui résoud tel problème pour résoudre ce même problème, on cherche à définir une certaine relation d'ordre entre les oracles, et à chercher, pour un problème donné, l'oracle minimal (s'il existe) permettant de le résoudre.

Une des contributions principales de cette thèse est d'avoir élaboré un cadre formel adéquat qui permette de définir et de comparer des oracles entre eux dans le cadre des protocoles de population. Ce cadre formel commence par remarquer que l'on peut définir les notions d'historique d'entrée et d'historique de sortie sans faire référence à une exécution d'un protocole. Ce sont simplement des ordonnancements de rencontres annotés par des valeurs (d'entrée ou de sortie).

Nous définissons alors un *comportement* B comme étant une relation entre historiques d'entrée et historiques de sortie. Autrement dit, étant donné un historique d'entrée H_{in}, le comportement B lui associe un ensemble $B(H_{in})$ d'historique de sortie. La notion de comportement est une façon de considérer le lien entre historiques d'entrée et historiques de sortie sans faire référence au mécanisme intérieur qui pourrait l'incarner. Ainsi, oracles et spécifications de problème sont modélisés par des comportements.

Étant donné un protocole de population $\mathcal{A}$, et un certain contexte (graphe de communication, hypothèse d'équité, etc.), on peut lui associer un comportement $Beh(\mathcal{A})$ obtenu en considérant les exécutions de A et uniquement celles-là. Autrement dit, $Beh(\mathcal{A})$ "oublie" le mécanisme interne (les règles du protocoles) et ne s'intéresse qu'aux entrées et sorties du protocole.

On peut également "composer" les comportements entre eux. Par exemple, si les alphabets correspondent, on peut diriger la sortie d'un comportement en entrée d'un autre comportement. Ou bien, on peut "paralléliser" deux comportements. À partir de là, la définition d'une relation d'ordre entre comportements se fait comme suit. On dit qu'un comportement B_1 est plus faible (au sens large) qu'un comportement B_2 s'il existe un protocole $\mathcal{A}$ et une certaine composition C des comportements $Beh(\mathcal{A})$ et B_2, de sorte que C soit compatible avec B_1 ; c'est-à-dire, quelque soit l'historique d'entrée H_{in}, on a la relation $C(H_{in}) \subseteq B_1(H_{in})$ entre ensemble d'historiques de sortie. Autrement dit, il s'agit de réduire le comportement B_2 au comportement B_1 par l'intermédiaire d'un protocole de population. Cette définition est analogue à la réduction de Karp en théorie de la complexité : un problème P_1 est plus faible qu'un problème P_2 s'il existe un algoritme en temps polynomial utilisant P_2 pour résoudre P_1.

2.2 Collecte de données

Notre étude des protocoles de population s'est porté en premier lieu au problème de la collecte de données qui s'énonce comme suit. Il existe, parmi les agents, un agent distingué, la station de base (BS), qui a des ressources illimitées. Les autres agents ont, initialement, des valeurs (par exemple, des résultats de mesure de capteurs), et le but consiste à transférer toutes ces valeurs vers la station de base.

Dans [4], les auteurs introduisent une nouvelle hypothèse d'équité sur les ordonnancements ; les *temps de couverture*. Cette hypothèse d'équité définit une sorte de synchronisme partiel dans l'ordonnancement des rencontres. Chaque agent x se voit attribuer un coefficient entier cv_x (son temps de couverture). Un ordonnancement S de rencontres est considéré comme équitable au sens des temps de couverture si quel que soit l'agent x, quel que soit le segment u de longueur cv_x dans S, x rencontre tous les autres agents au moins une fois durant u. Ainsi, un agent avec un faible temps de couverture est, d'une certaine façon, un agent rapide. Cette hypothèse de synchronisme relatif est dite partielle car les temps de couverture ne sont pas supposés connus des agents.

L'avantage de cette hypothèse de temps de couverture est de permettre une évaluation du temps de convergence de protocoles ; dans le cas de collecte de

données, le temps de collecte de toutes les valeurs initalement présentes. Dans [4], les auteurs ont élaboré un protocole $TTFM$ (Transfer To the Fastest Marked) qui résoud ce problème de manière optimale : le temps de convergence de $TTFM$ atteint la borne minimale. Cependant, $TTFM$ suppose que quand deux agents se rencontrent, ils peuvent déterminer lequel a le temps de couverture le plus élevé.

Nous avons appliqué cette hypothèse à l'étude d'un protocole de collecte de données appelé ZebraNet. ZebraNet est un projet mené par l'Université de Princeton et déployé au Kenya. Il étudie des populations de zèbres en les équipant de capteurs. Le protocole ZebraNet fonctionne de la manière suivante : les agents (zèbres) qui rencontrent plus souvent la station de base sont estimés plus rapides par les autres agents ; lorsque deux agents se rencontrent, celui qui estime être le plus rapide des deux récupère les valeurs de l'autre (dans la limite de mémoire disponible), et cet autre agent efface ses valeurs (il considère qu'il a transféré ses valeurs). Ainsi la principale différence avec $TTFM$ est que les agents ne peuvent pas comparer leurs temps de couverture, et tentent d'estimer leurs vitesses relatives en comptant le nombre de rencontres avec la station de base.

La table 2.2 résume nos contributions sur ce sujet. Nous avons d'abord montré que le protocole ZebraNet original ne converge pas toujours (temps de convergence infini). Nous avons alors proposé deux versions modifiées de ZebraNet, $MZP1$ et $MZP2$, qui convergent toujours, et nous avons calculé leurs temps de convergence exacts.

TTFM	ZP	MZP1	MZP2
$2 \cdot \min(cv_x)$	∞	$\sum cv_x$	$2 \cdot \max(cv_x)$

On voit que, grâce à l'introduction d'une hypothèse d'équité suffisamment forte dans le modèle des protocoles de population, une analyse de protocole concret est possible.

2.3 Consensus

Le problème du consensus est un problème classique en algoritmique répartie. Chaque agent possède une valeur initiale et doit prendre une décision irréversible au bout d'un certain temps (terminaison). De plus, les agents décident la même valeur (accord), et celle-ci doit figurer parmi les valeurs initiales des agents (validité). Nous avons étudié ce problème dans le cadre de graphes de communication complets, en se fondant sur une hypothèse d'équité classique (chaque agent rencontre infiniment souvent chacun des autres agents). Notez qu'ici, toutes les exécutions considérées n'admettent aucune défaillance.

Nous avons montré que, sans oracle, ce problème était impossible à résoudre. La preuve repose sur une technique connue de partitionnement du graphe de communication. Pour contourner ce résultat, nous avons introduit une classe

d'oracles (comportements) appelée $Mnemosyne$. Un oracle de $Mnemosyne$ ne fait qu'observer le passé causal (dans l'ordonnancement des rencontre) des agents et renseigne sur la présence ou l'absence de certaines séquences typiques de rencontres. Nous avons défini un oracle particulier, $DejaVu$, dans cette classe, et nous avons construit un protocole qui résoud le consensus à l'aide de cet oracle.

Notre contribution la plus importante, sur ce sujet, a été de montrer que l'oracle $DejaVu$ est l'oracle minimal dans la classe $Mnemosyne$ permettant de résoudre le consensus *symétrique* ; variante du consensus dans laquelle on exige, en plus des conditions ci-dessus, que la valeur de décision soit indépendante de la distribution des valeurs initiales dans la population.

2.4 Élection de leader (autostabilisante)

L'élection de leader, comme le consensus, est un problème fondamental en algoritmique répartie. Il s'agit principalement de sélectionner un unique agent dans la population de manière permanente. Dans un premier temps, nous avons étudié ce problème dans le cas où les agents de la population sont correctement initialisés (on suppose une initialisation identique pour tous les agents afin d'éviter les solutions triviales). Puis dans un deuxième temps, nous avons introduit la possibilité de défaillances transitoires, et avons cherché à construire des solutions *auto-stabilisantes*. Une défaillance transitoire est une défaillance ponctuelle qui peut corrompre l'état (la mémoire) d'un sous-ensemble quelconque des agents. Un système auto-stabilisant (notion introduite par Dijkstra [8]) est un système qui tolère ce type de défaillance dans le sens suivant : après la dernière défaillance transitoire, il existe un temps fini après lequel le système se comporte de manière correcte. Cela revient à dire que, dans toute exécution sans défaillance à partir d'une configuration *arbitraire*, il y a un suffixe de l'exécution qui est correct.

Ainsi, notre étude de ce problème se scinde en deux parties ; la première traitant du cas avec initialisation uniforme, la seconde avec initialisation arbitraire.

(Initialisation uniforme). Lorsque l'initialisation est uniforme, le problème de l'élection de leader est essentiellement une affaire de brisure de symétrie. Reprenant des concepts de [1,5], nous avons montré qu'une hypothèse d'équité trop faible (e.g., ne requérir que le fait que les arcs du graphe de communications soient sélectionnés infiniment souvent par l'ordonnanceur) empêche l'existence d'une solution. Plus précisément, sur un graphe G qui est un revêtement (strict) d'un autre graphe, quelque soit le protocole $\mathcal{A}$ considéré, il existe une exécution (équitable) de $\mathcal{A}$ sur G telle qu'apparaisse infiniment souvent une configuration dans laquelle l'état d'un agent est toujours partagé par au moins un autre agent. Si on prétendait que $\mathcal{A}$ fusse une solution au problème d'élection de leader, alors des configurations avec deux leaders au moins apparaitraient infiniment souvent dans cette exécution.

Cependant, pour briser cette symétrie, on peut utiliser une hypothèse d'équité plus proche d'une forme d'aléatoire. Cette hypothèse d'équité, connue sous le nom d'équité globale (global fairness), a été introduite dans [2], et garantit la

chose suivante : si une configuration du protocole est accessible infiniment souvent, alors cette configuration est atteinte infiniment souvent. Cette propriété est analogue à une propriété des séquences binaires aléatoires disant que, puisque telle séquence finie a une probabilité non-nulle d'apparaître, celle-ci apparaît dans la séquence infinie avec probabilité un.

Grâce à cette hypothèse d'équité globale, nous avons construit un protocole permettant de résoudre l'élection de leader sur toute la famille des graphes quelconques (connectés).

(Initialisation arbitraire). L'une des difficultés lorsque la configuration initiale est quelconque est qu'il est possible qu'il n'y ait aucun leader dans celle-ci. Ainsi un protocole auto-stabilisant permettant de résoudre l'élection de leader doit nécessairement comporter un mécanisme de création de leader. Or, l'hypothèse d'équité globale fait que s'il est possible de créer un leader infiniment souvent, alors un leader sera créé. En combinant cette propriété à un argument de partition, les auteurs de [3] ont montré qu'il n'existe aucun protocole auto-stabilisant capable de résoudre l'élection de leader sur la famille des graphes complets, et plus généralement sur une famille non-simple [1] de graphes.

Pour contourner ce résultat, les auteurs de [10] ont introduit (de manière informelle) un oracle noté $\Omega?$ qui fonctionne de la manière suivante. Contrairement aux oracles de la classe $Mnemosyne$, l'oracle $\Omega?$ ne se contente pas d'observer les rencontres passées, mais observe également un bit des états des agents ; il a accès aux configurations. Chez ces auteurs, le bit observé est associé au statut de leader ou non-leader de l'agent. Les sorties de $\Omega?$ renseignent alors sur la présence ou l'absence de leader dans le système. Si il n'y a, de manière permanente, aucun leader dans la population, alors $\Omega?$ sortira la valeur 0 à au moins un des agents. Si il y a, de manière permanente, au moins un leader dans la population, alors $\Omega?$ sortira la valeur 1 à tous les agents de la population. Grâce à cet oracle, les auteurs de [10] ont construit des solutions autostabilisantes pour la famille des graphes complets et pour la famille des anneaux.

En utilisant le même oracle $\Omega?$, nous avons généralisé leurs résultats en construisant, pour chaque entier δ, un protocole autostabilisant $\mathcal{A}_\delta$ qui résout l'élection de leader sur la famille des graphes de degrés inférieurs ou égaux à δ. L'idée consiste à créer un leader lorsque $\Omega?$ sort la valeur 0. Chaque leader essaie de "tuer" les autres leaders éventuels en émettant des jetons qui circulent dans la population. Cependant, pour éviter que tous les leaders disparaissent, un mécanisme de barrières circulantes est également mis en place. Le fait que le degré soit borné implique d'une certaine façon qu'il n'y a pas "trop" de chemins dans le graphe, ce qui fait qu'avec un nombre fini d'états (dépendant uniquement de d), le protocole est capable de contrôler la circulation des jetons et des barrières.

La question se pose alors de savoir si on peut construire un protocole autostabilisant qui résolve l'élection de leader sur la famille des graphes quelconques (sans condition sur le degré). L'obstacle dans ce cas-là est qu'il y a trop de

1. Une famille $\mathcal{F}$ est non simple s'il existe $G, H_1, H_2 \in \mathcal{F}$ tels que H_1 et H_2 soient des sous-graphes disjoints de G.

chemins, et l'idée du protocole précédent ne s'applique plus (le nombre d'états dépendrait de la taille de la population).

Pour contourner cet obstacle, nous avons proposé une généralisation de l'oracle $\Omega?$. Un entier d étant fixé, l'oracle $\Omega?(d)$ donne une estimation entre $0, 1, \ldots, d$ du nombre de leaders dans la population. Ainsi, si il n'y a jamais aucun leader durant l'exécution, l'oracle $\Omega?(d)$ sortira 0 à quelque agent. S'il y a toujours entre 1 et 4 leaders (par exemple), alors l'oracle $\Omega?(d)$ sortira des valeurs entre 1 et 4 à tous les agents. Le cas $d = 1$ correspond à l'oracle $\Omega?$. Nous avons alors construit un protocole autostabilisant qui résout l'élection de leader à l'aide de l'oracle $\Omega?(2)$ sur la famille des graphes quelconques.

Nous avons également construit une solution selon une autre approche. L'oracle $\Omega? \otimes \Omega?$ représente la mise en parallèle de deux instances de l'oracle $\Omega?$. Étant donné que la difficulté du passage des graphes de degrés bornés aux graphes arbitraires réside dans la difficulté de contrôler la circulation des jetons (éviter de tuer tous les leaders), nous utilisons une instance de $\Omega?$ pour contrôler la création de leader comme précédemment, et une instance de $\Omega?$ pour contrôler la création de jeton. Nous avons pu ainsi construire un autre protocole autostabilisant résolvant l'élection de leader sur la famille des graphes arbitraires avec l'oracle $\Omega? \otimes \Omega?$.

Enfin, nous avons estimé la force des oracles $\Omega?(d)^{\otimes k}$ relativement au problème de l'élection de leader autostabilisante. Nous avons montré que, sur la famille des graphes complets (plus généralement sur une famille non-simple de graphes), résoudre l'élection de leader autostabilisante ne permet pas d'implémenter l'oracle $\Omega?$ (a fortiori les oracles $\Omega?(d)^{\otimes k}$ non plus). Par contre, sur la famille (simple) des anneaux, nous avons montré que l'élection de leader autostabilisante et l'implémentation autostabilisante de l'oracle $\Omega?$ sont deux problèmes équivalents.

La table 2.4 résume nos contributions sur l'élection de leader. LE (Leader Election) dénote le problème de l'élection de leader. $SSLE$ (Self-Stabilizing Leader Election) dénote le problème de la recherche de protocole autostabilisant résolvant l'élection de leader. La relation d'ordre $A \preccurlyeq B$ est la relation de comparaison entre comportements, et signifie qu'on peut implémenter A à l'aide de B dans les conditions mentionnées.

Famille de graphes	Initialisation	Équité	Notes
Contient un revêtement	Uniforme	Classique	LE est impossible
Arbitraire	Uniforme	Globale	LE est possible
Degré borné	Arbitraire	Globale	$SSLE \preccurlyeq \Omega?$
Arbitraire	Arbitraire	Globale	$SSLE \preccurlyeq \Omega?(2)$
			$SSLE \preccurlyeq \Omega?^{\otimes 2}$
Anneaux	Arbitraire	Globale	$SSLE \simeq \Omega?$
Non-simple	Arbitraire	Globale	$SSLE^{\otimes k} \not\preccurlyeq \Omega?$

3 Réplication

Dans la deuxième partie de ce mémoire, nous avons étudié le problème de la réplication de machines finies dans le modèle classique de passage de message asynchrone, avec possibilité de pannes crash. Ce problème se pose, par exemple, lorsqu'on essaie de fournir un service à un certain nombre de clients. On met en place un système d'information, et les clients peuvent y accéder en envoyant des requêtes. L'idée principale consiste à dupliquer le programme central (la machine finie) sur plusieurs serveurs de sorte que si une partie du système tombe en panne, il est a priori possible de maintenir le service.

La difficulté réside dans le fait que ces différents serveurs, ou réplicas, doivent se coordonner pour ne pas donner des réponses incohérentes aux différents clients. En effet, dans un système réparti asynchrone, les requêtes des clients n'arrivent pas forcément en même temps, ni dans le même ordre aux différents réplicas. On requiert en général que l'ensemble des réplicas se comporte comme une unique machine finie qui traiterait les différentes requêtes séquentiellement (condition de linéarisabilité), et que cet ensemble de réplicas traite en temps fini toutes les requêtes reçues (condition de vivacité).

Ce problème est loin d'être trivial, et est intrinsèquement lié au problème du consensus. En effet, les réplicas doivent, d'une certaine façon, de se mettre d'accord sur la séquence des requêtes à exécuter. Si les réplicas démarrent dans le même état, et si, à chaque étape, ils utilisent un algoritme de consensus pour décider de la requête à exécuter, alors il est possible de résoudre le problème.

Or, dans un réseau à passage de message asynchrone susceptible de pannes crash (pannes définitives), il a été montré que le problème du consensus est insoluble [11]. En particulier, cela affecte la possibilité d'un protocole de réplication. Malgré tout, Lamport a élaboré un algoritme, nommé Paxos [12,13], qui résoud partiellement ce problème. En effet, Paxos garantit, en toutes circonstances, la condition de linéarisabilité, i.e., les réplicas ne peuvent pas donner de réponses incohérentes. Par contre, la condition de vivacité, c'est-à-dire, le fait que des requêtes vont effectivement être traitées, n'est garantie que dans certaines conditions ; savoir, l'existence d'un leader unique et stable. En pratique garantir la stabilité d'un unique leader est faisable, ce qui justifie le déploiement de Paxos au niveau industriel.

3.1 Autostabilisation pragmatique

Aucune des approches mentionnées ci-dessus ne considèrent la question des défaillances transitoires, c'est-à-dire, de défaillances ponctuelles pouvant corrompre l'intégrité de la mémoire des processeurs et mettre le système dans une configuration arbitraire. Dans le contexte de la réplication de machine finie, ces défaillances peuvent induire deux types d'effets. D'abord, elles peuvent corrompre l'état des copies du programme en chaque réplica. Ce faisant, même si les réplicas exécutent les mêmes requêtes dans le même ordre, puisqu'elles ne partent pas du même état, elles risquent de donner des réponses incohérentes, et violer la condition de linéarisabilité. Ce n'est pourtant pas le point le plus

important, car si les réplicas ont toujours la possibilité de se mettre d'accord sur quelque chose, ils peuvent se mettre d'accord sur un certain état initial. Par contre, si une défaillance transitoire affecte la capacité même des réplicas à se mettre d'accord, alors la situation est plus critique ; il n'y a plus aucun moyen de réparer le système [2].

Une des caractéristiques de l'autostabilisation est qu'elle ne concerne que les problèmes "vivaces", c'est-à-dire, les problèmes dans lesquels les processeurs doivent garantir un service permanent. En l'occurrence, parler de d'algoritme autostabilisant qui résoud le consensus n'a pas de sens ; une défaillance transitoire peut toujours forcer les processeurs à décider dès le départ des valeurs incohérentes. Ceci dit, le problème de la réplication de machine finie est un problème vivace. La seule conséquence de la remarque précédente est qu'on ne peut pas garantir que partant d'une configuration arbitraire les réplicas exécutent d'emblée la même séquence de requêtes à partir du même état initial.

Un des principaux ingrédients d'un protocole de réplication fondé sur Paxos consiste en la capacité de distinguer les messages récents et les messages anciens. D'un point de vue très abstrait, on peut utiliser des entiers non-bornés pour estampiller les messages ; ce qui suppose une mémoire infinie. À un niveau plus concret, les processeurs ont une mémoire finie, et on utilise alors des entiers bornés par une certaine valeur finie $2^{\mathfrak{b}}$, où $\mathfrak{b}$ est la taille en bits des registres. Intuitivement, cela signifie que l'algoritme original de Paxos est capable de discerner des messages dans une fenêtre temporelle de l'ordre de $2^{\mathfrak{b}}$.

Cette constante est tellement large qu'elle suffit amplement pour la plupart des applications concrètes ; du moins, tant que les défaillances transitoires ne sont pas prises en compte. En effet, en incrémentant un compteur de 64 bits initialisé à zéro toutes les nanosecondes, il faudrait cinq cents ans avant d'atteindre la valeur maximale 2^{64}. Cependant une défaillance transitoire peut corrompre la valeur des compteurs et les ajuster à une valeur proche de la borne. Ceci peut compromettre de manière définitive la linéarisabilité et la vivacité de Paxos.

Cette dernière remarque conduit à la formulation légèrement plus faible de la notion de système autostabilisant. Dans l'autostabilisation classique, on cherche à montrer qu'une exécution (infinie) partant d'une configuration arbitraire admet un suffixe (infini) durant lequel le système satisfait les spécifications du problème. Nous définissons la notion d'autostabilisation pragmatique : on cherche simplement à montrer qu'une exécution (infinie) partant d'une configuration arbitraire contient un segment (fini) durant lequel le système satisfait les spécifications du problème. On ajoute cependant que ce segment doit être suffisamment long par rapport une échelle temporelle prédéfinie. Plus précisément, on exige que ce segment contienne une chaine causale de longueur supérieure ou égale à $2^{\mathfrak{b}}$. Un algoritme qui satisfait cette propriété est dit pragmatiquement autostabilisant.

Cette propriété peut paraître étrange au premier abord. Mais on doit remarquer qu'une implémentation de Paxos (avec initialisation) qui se fonderait sur des compteurs entiers bornés, ne peut fonctionner correctement que pendant une durée de l'ordre de $2^{\mathfrak{b}}$.

2. mis à part un redémarrage global, ce qui ne constitue pas une solution répartie.

3.2 Contribution

Comme dit précédemment, un des mécanismes essentiels de Paxos réside dans l'utilisation intelligente d'estampilles ; ces estampilles étant, dans la version originale, de simples entiers (non-bornés dans la version abstraite, bornés en pratique).

Notre principale contribution consiste en la définition d'un nouveau système d'estampilles ainsi que son insertion au coeur de Paxos. Nous avons alors montré que, quelles que soient les conditions d'exécutions, la partie de l'algoritme liée au système d'estampilles converge toujours, et atteint un segment pratiquement infini (longueur de l'ordre de 2^b) durant lequel le système d'estampilles est correct. Nous avons ensuite montré que durant ce segment d'exécution, l'algoritme complet satisfait les mêmes propriétés que Paxos : la linéarisabilité est assurée, mais la vivacité n'est assurée qu'à condition qu'un unique leader soit actif.

Références

1. D. Angluin. Local and global properties in networks of processors. In *12th Symposium on the Theory of Computing*, pages 82–93. ACM, 1980.
2. D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4) :235–253, 2006.
3. D. Angluin, J. Aspnes, M. J. Fischer, and H. Jiang. Self-stabilizing population protocols. In *OPODIS*, pages 103–117, 2005.
4. J. Beauquier, J. Burman, J. Clement, and S. Kutten. On utilizing speed in networks of mobile agents. In *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing, PODC 2010, Zurich, Switzerland, July 25-28, 2010*, pages 305–314. ACM, 2010.
5. P. Boldi, S. Shammah, S. Vigna, B. Codenotti, P. Gemmell, and J. Simon. Symmetry breaking in anonymous networks : Characterizations. In *ISTCS*, pages 16–26, 1996.
6. T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2) :225–267, 1996.
7. B. Charron-Bost, M. Hutle, and J. Widder. In search of lost time. *Inf. Process. Lett.*, 110(21) :928–933, 2010.
8. E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. of the ACM*, 17(11) :643–644, Nov. 1974.
9. C. Dwork, N. A. Lynch, and L. J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2) :288–323, 1988.
10. M. Fischer and H. Jiang. Self-stabilizing leader election in networks of finite-state anonymous agents. In *OPODIS*, pages 395–409, 2006.
11. M. H. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of consensus with one faulty process. *Journal of the ACM*, 32(2) :374–382, Apr. 1985.
12. L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2) :133–169, May 1998.
13. L. Lamport. Paxos made simple. *ACM SIGACT News*, 32(4) :18–25, Dec. 2001.