

Agreement without knowing everybody: a first step to dynamicity

Mohssen Abboud
LIAFA- CNRS et Université
Paris 7, Case 7014
75205 Paris Cedex 13, France
maboud@liafa.jussieu.fr

Carole Delporte-Gallet
LIAFA- CNRS et Université
Paris 7, Case 7014
75205 Paris Cedex 13, France
cd@liafa.jussieu.fr

Hugues Fauconnier
LIAFA- CNRS et Université
Paris 7, Case 7014
75205 Paris Cedex 13, France
hf@liafa.jussieu.fr

ABSTRACT

We study in this paper the consensus problem in asynchronous models where the set of participating processes is not known. We prove that in this model the consensus is impossible to be solved even if no process may crash. We prove that the asynchronous model augmented with failure detector Σ , which enables to realize a quorum, is sufficient to circumvent this impossibility result. For this, we present an algorithm solving the consensus problem in this model.

Categories and Subject Descriptors

C.2.4 [Computer-communication Networks]: Distributed Systems, distributed application, distributed databases, network operating systems; D.4.5 [Operating Systems]: Reliability, fault tolerance; F.1.1 [Computation by Abstract Devices]: Models of Computation; H.2.4 [Database Management]: System concurrency, distributed databases

General Terms

Algorithms, Reliability, Theory

Keywords

Distributed Algorithms, Fault Tolerance, Dynamic Systems

1. INTRODUCTION

Le problème du consensus est un paradigme essentiel de l'algorithmique distribuée tolérante aux pannes: la plupart des problèmes de cohérence peuvent se réduire au consensus [8]. Il est donc indispensable de pouvoir trouver des solutions à ce problème dans un environnement distribué. Pourtant un résultat classique [6] montre qu'il est impossible à résoudre en présence de pannes dès que la communication est asynchrone. Aussi de nombreux travaux de l'algorithmique distribuée tolérante aux pannes sont consacrés à contourner ce résultat d'impossibilité. Une des solutions pour cela consiste à introduire des *détecteurs de défaillances* [2] qui don-

nent des informations qui ne sont pas toujours fiables aux processus sur les pannes. C'est l'approche qui est suivie ici.

Cependant le contexte est ici légèrement différent du contexte habituel, et nous allons donc utiliser des détecteurs de défaillances dans un autre contexte : celui où on ne connaît pas de borne sur l'ensemble des processus participants.

En général, les solutions concernant le maintien de la cohérence dans des environnements distribués supposent que l'ensemble des processus participants dans le système est connu ou au moins qu'une borne sur leur nombre est connue. Ainsi de nombreux protocoles supposent d'attendre de recevoir un "ack" d'une majorité des participants ce qui suppose d'en connaître le nombre.

Ces hypothèses sont naturelles pour des réseaux locaux, mais elles ne sont pas réalistes dès que l'on considère des réseaux à plus large échelle ou des réseaux dynamiques. Par exemple dans un système pair à pair, le nombre de participants peut être arbitrairement grand (potentiellement tous les sites Internet peuvent y participer) et dans un système dynamique le nombre de participants change avec le temps.

Dans cet article, nous allons nous intéresser au problème du consensus dans des systèmes asynchrones dans le cas où l'ensemble des processus participants n'est pas connu mais où chaque processus a une identité unique.

Pour cela il faut préciser le modèle de défaillances: un processus qui peut, potentiellement, participer et qui ne participe pas au système est-il défaillant? En le considérant comme défaillants on retrouve le modèle des "initially dead" [6]. Dans la suite, on supposera pour simplifier que une fois créé un processus reste vivant pour toujours.

D'une certaine façon les résultats présentés ici montrent que même avec cette hypothèse très faible sur les défaillances, le consensus est impossible à résoudre sans la connaissance des participants. Cependant, on montre ensuite que si les processus disposent d'informations supplémentaires qui permettent d'obtenir un quorum (ces informations correspondent à celles fournies par le détecteur de défaillances Σ [3, 4]), alors on peut résoudre le problème du consensus. Pour pouvoir réaliser le détecteur de défaillances Σ on montre que la connaissance d'un nombre m tel que si n est le nombre de processus vivants dans le système on a $n \geq m \geq \lfloor n/2 \rfloor + 1$ permet de réaliser ce détecteur de défaillances.

Le reste de l'article est organisé comme suit: dans un premier temps nous définissons le modèle (modèle de processus et modèle de communications). Dans la section suivante, nous montrons le principal résultat d'impossibilité, puis nous donnons la définition formelle du détecteur de défaillances Σ

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CDUR-NOTERE 2008 June 23-27, 2008, Lyon, France

Copyright 2008 ACM 978-1-59593-937-1/08/0003 ...\$5.00.

utilisé pour contourner cette impossibilité, et nous présentons et prouvons un algorithme qui utilise ce détecteur de défaillances pour résoudre le problème consensus.

2. MODÈLE

On définit dans cette section le système \mathcal{S} que l'on considère.

2.1 Modèle de processus

Chaque processus a une identité unique prise dans un ensemble $\Pi = \{\dots, i, j, k, \dots\}$, qui représente l'ensemble des identités possibles des processus. Dans la suite un processus sera représenté par son identité. L'ensemble Π peut être infini,¹ mais on suppose que dans chaque exécution au plus un nombre fini de processus est créé. Si un processus est créé à l'instant t , à partir de l'instant t il est vivant. Pour une exécution donnée, l'ensemble des processus vivants noté Viv est simplement l'ensemble des processus participants.

Pour simplifier on supposera que le temps \mathcal{T} est discret et n'est pas connu des processus, à divers instants t des processus exécutent des pas de calculs atomiques. Les processus sont asynchrones: le délai entre deux pas de calcul de processus peut être arbitrairement grand mais est fini. On suppose aussi que les processus vivants font une infinité de pas de calculs.

2.2 Modèle de communication

Les processus communiquent par envoi et réception de messages. On suppose que le graphe de communication est complet: il existe un canal de communication entre tout p et tout q . On suppose de plus que les messages émis ne sont ni altérés ni dupliqués (propriété d'intégrité). D'une manière générale un message émis par p à l'instant t vers q sera reçu par un processus q vivant à un instant $t' \geq t$ (on suppose donc qu'il n'y a pas de pertes de messages).

On suppose l'existence d'une primitive de diffusion fiable qui permet à un processus p d'envoyer un message m à tous les processus vivants. La diffusion d'un message m se fait par la primitive $diff(m)$ et la réception du message m par $receive(m)$. Plus précisément, si à l'instant t , p fait un $diff(m)$ alors pour tout processus vivant q il existe $t' \geq t$ tel que q recevra le message m à l'instant t' . Le délai entre l'envoi et la réception du message peut être arbitraire.

Notons qu'on peut sans difficulté supposer qu'au lieu d'un graphe complet de communication on dispose seulement d'un graphe de communication fortement connexe dans lequel chaque processus ne peut envoyer directement un message qu'à ses voisins immédiats. Pour cela, pour envoyer un message m à tous les processus, un processus envoie m à tous ses voisins et tous les processus relaient à leurs voisins une fois au plus chaque message m .² De cette façon un processus n'a besoin de connaître que l'identité de ses voisins. Notons aussi que, du fait de l'asynchronisme des communications, un processus ne peut pas déterminer l'ensemble de tous les processus vivants.

3. CONSENSUS

Le problème du consensus est un problème classique fondamental de l'algorithmique distribuée tolérante aux pannes

¹On suppose dans la suite que Π contient au moins deux processus.

²On ne s'intéresse pas ici à l'efficacité de cette diffusion.

et de nombreux problèmes d'accord et de cohérence peuvent se ramener au problème du consensus.

Un algorithme de consensus est un algorithme de décision pour lequel on suppose que chaque processus p dispose d'une valeur initiale v_p , les décisions des processus sont irrévocables et doivent assurer les propriétés suivantes:

- *Terminaison*: tout processus participant doit décider d'une valeur,
- *Accord*: si p et q décident ils doivent décider la même valeur,
- *Validité*: si p décide la valeur v alors v est la valeur initiale d'un processus participant.

Nous dirons qu'un algorithme résout le consensus pour un ensemble E , s'il résout le problème du consensus pour tout ensemble de participants inclus dans E .

Rappelons tout d'abord qu'il n'existe pas de solution au problème du consensus tolérant même une seule panne dans le cas où la communication est asynchrone (même si le nombre de processus est connu) [6].

Nous allons dans la section suivante étendre le résultat d'impossibilité du consensus au modèle que nous considérons ici.

3.1 Impossibilité du consensus

Montrons:

THÉORÈME 1. *Dans \mathcal{S} , il n'existe pas d'algorithme de consensus pour un ensemble E dès que E a plus de deux éléments.*

PREUVE. Supposons qu'il existe un algorithme A , qui résout le consensus. Comme on a supposé que E contient au moins deux éléments, on peut partitionner E en deux sous-ensembles disjoints E_1 et E_2 .

Considérons une première exécution e_1 dans laquelle la valeur initiale des processus est v_1 et l'ensemble V_1 des processus vivants est inclus dans E_1 , alors, d'après la validité et la terminaison du consensus il existe un temps t_1 après lequel tous les processus vivants ont décidé v_1 .

Considérons une deuxième exécution e_2 dans laquelle la valeur initiale des processus est v_2 et l'ensemble V_2 des processus vivants est inclus dans E_2 , alors, d'après la validité et la terminaison du consensus il existe un temps t_2 après lequel tous les processus vivants ont décidé v_2 .

Considérons maintenant une troisième exécution dans laquelle l'ensemble des processus vivants est égale à l'union de V_1 et de V_2 . Les processus de V_1 ont v_1 comme valeur initiale et ceux de V_2 ont v_2 comme valeur initiale. Dans cette exécution, tous les messages entre processus de V_1 et processus de V_2 sont retardés jusqu'à après le temps $\max(t_1, t_2)$, les messages entre processus de V_1 se comportent comme dans e_1 et les messages entre les processus de V_2 se comportent comme dans e_2 . Cette exécution est indistinguable de e_1 pour les processus de V_1 jusqu'au temps t_1 et est indistinguable de e_2 pour les processus de V_2 jusqu'au temps t_2 . On en déduit que les processus de V_1 ont décidé v_1 au temps t_1 et que les processus de V_2 ont décidé v_2 au temps t_2 . Ce qui contredit la propriété d'accord du consensus. \square

3.2 Détecteur de défaillances Σ

Dans un système asynchrone avec des défaillances de processus, de nombreux problèmes comme le consensus ne peuvent être résolus. Les détecteurs de défaillances [2] sont des oracles distribués que les processus peuvent consulter et qui donnent des informations sur les défaillances des processus. Grâce à eux, on peut résoudre des problèmes qui, sans eux, n'auraient pas de solution. Les détecteurs de défaillances encapsulent les informations nécessaires sur les défaillances des processus.

Dans la suite on considérera une extension du détecteur de défaillances Σ . On peut trouver la définition formelle de ce dernier dans [3]. De façon informelle, dans un système où les processus peuvent tomber en panne, le détecteur Σ est un détecteur de défaillances qui retourne des listes de processus supposés être corrects qui réalisent un quorum. Plus précisément, la sortie de Σ pour un processus p à un instant est une liste de processus en qui p a confiance, et l'intersection de ces listes pour deux processus n'est jamais vide, et, de plus, ultimement chaque liste ne contient que des processus corrects.

Nous allons adapter la définition de Σ au modèle considéré ici où il n'y a pas à proprement parler de processus défaillants: ici, un processus correct est simplement un processus vivant. On suppose donc que les processus peuvent à tout moment interroger leur détecteur de défaillances Σ , qui retourne une liste de processus. Si pour une exécution donnée $\Sigma(p, t)$ est le résultat de l'interrogation de Σ par le processus p à l'instant t , on a les propriétés suivantes:

- Intersection : deux listes contiennent toujours au moins un processus commun.
 $\forall p, q \in Viv, \forall t, t' \in \mathcal{T} :$
 $\Sigma(p, t) \cap \Sigma(q, t') \neq \emptyset;$
- Complétude: de façon ultime, les listes retournées ne contiennent que des processus vivants.
 $\forall p \in Viv, \exists t \in \mathcal{T}, \forall t' \in \mathcal{T} : t' > t : \Sigma(p, t') \subseteq Viv.$

Remarque: En supposant que le nombre de processus participants est n , si on ne connaît qu'une estimation m de ce nombre, on peut montrer que si $\lfloor n/2 \rfloor + 1 \leq m \leq n$ on peut alors implémenter Σ . Pour cela, chaque processus envoie à tous un message avec son identité indiquant qu'il est vivant. Initialement la liste sortie pour réaliser l'implémentation du détecteur de défaillances est Π et dès qu'un processus a reçu m identités différentes, ces m identités constituent la liste sortie pour réaliser le détecteur de défaillances. Comme $m \leq n$, il existera bien un temps à partir duquel chaque processus vivant aura bien une liste de m identités de processus vivants. Le fait que $m \geq \lfloor n/2 \rfloor + 1$ entraîne la propriété d'intersection.

3.3 Consensus en utilisant le détecteur de défaillances Σ

En supposant que l'on dispose du détecteur de défaillances Σ , nous allons montrer:

PROPOSITION 1. *Dans \mathcal{S} avec le détecteur de défaillances Σ , il est possible de résoudre le problème du consensus.*

Pour cela, nous adaptons l'algorithme classique [6]. L'algorithme de la Figure 1 implémente un consensus dans un système asynchrone sans défaillances.

Principe de fonctionnement: Chaque processus p va construire un graphe (V_p, E_p) . V_p est l'ensemble des sommets, c'est l'ensemble des processus du système dont p a connaissance directement ou indirectement. C'est donc un sous ensemble de Viv . E_p est l'ensemble des arcs, un arc de x à y indique que x a eu connaissance directement de y .

Pour se faire, chaque processus p diffuse un message de type 1, contenant son identité, puis il attend de recevoir un message de type 1 d'au moins tous les processus contenus dans une des sorties de Σ . La consultation du détecteur de défaillances se fait par $QueryFD_{\Sigma}()$. Le processus p inclut dans V_p les processus dont il a reçu un message et inclut dans E_p un arc vers chacun de ceux-ci. La propriété de complétude de Σ assure que p recevra un message d'au moins tous les processus contenus dans une des sorties de Σ .

Dans un deuxième temps chaque processus p diffuse un message de type 2 contenant son identité, sa liste V_p de processus dont il a eu connaissance directement ainsi que v_p , sa valeur proposée au consensus. Sur réception d'un message $(2, q, V', e)$ d'un processus q , p met à jour son graphe: il ajoute à V_p l'ensemble V' , et à E_p les arcs de q vers les sommets de V' . Il enregistre par ailleurs la valeur proposée par q dans le tableau EST . Il attend ainsi de recevoir des messages jusqu'à avoir complété le graphe, i.e. d'avoir reçu un message de tous les sommets de V_p . Comme tous les processus qui ont envoyé un premier message enverront un deuxième message, l'attente de p est fini.

Deux processus peuvent ne pas avoir le même graphe. Mais on montre que, par les propriétés de Σ , tous les graphes contiennent une composante fortement connexe puits³ P , et cette composante est commune à tous les processus. Chaque processus extrait P et choisit comme valeur du consensus la plus petite valeur proposée par les processus de P .

DÉFINITION 1. *Pour un processus p , W_p est l'ensemble des sommets obtenus quand l'itération ligne 12 termine pour p . $G_p = (V_p, E_p)$ est le graphe construit par p par l'algorithme de la Figure 1.*

LEMME 2. *Pour tout processus, le graphe obtenu par l'algorithme de la Figure 1 pour ce processus possède une seule composante fortement connexe puits.*

PREUVE. Soit p un processus, par construction il y a un chemin de p à tous les sommets de V_p . Donc G_p est connexe. Comme G_p est connexe, il y a dans son graphe des composantes fortement connexes, un sommet source (qui n'a aucun arc entrant) et un sommet puits (qui n'a aucun arc sortant). \square

LEMME 3. *Pour tout processus p , si un processus z appartient à V_p alors $W_z \subseteq V_p$ et pour tout processus x de W_z (z, x) appartient à E_p et pour tout processus y n'appartenant pas à W_z (z, y) n'appartient pas à E_p .*

PREUVE. Soit p un processus et z un processus de V_p .

Si $p = z$, après la ligne 12 V_p contient W_p et E_p contient tous les arcs entre p et un élément de V_p . Ensuite E_p et V_p ne font que croître. De plus aucun arc issu de p ne sera ajouté ensuite dans la deuxième itération.

³Une composante fortement connexe P d'un graphe (V, E) est un puits si: $\forall v \in P, \forall v' \in V \setminus P (v, v') \notin E$.

```

Code de  $p$ :
Initialisation:
1  $\forall q \ EST[q] \leftarrow \perp$ 
2  $EST[p] \leftarrow v_p$ 
3  $Vu \leftarrow \{p\}$ 
4  $V \leftarrow \{p\}$  *Sommets du graphe*
5  $E \leftarrow \emptyset$  *Arcs du graphe*

6 diff(1,  $p$ )
7 repeat
8    $A \leftarrow QueryFD_{\Sigma}()$ 
9   if received (1,  $q$ ) from some  $q$  then
10      $V \leftarrow V \cup \{q\}$ 
11      $E \leftarrow E \cup \{(p, q)\}$ 
12 until  $A \cup \{p\} \subseteq V$ 

13 diff(2,  $p, V, v_p$ )
14 repeat
15   wait until received(2,  $q, *, *$ ) with  $q \in V \setminus \{p\}$ 
16   let (2,  $q, V', e$ ) such a message
17    $E \leftarrow E \cup \{(q, q') | q' \in V'\}$ 
18    $V \leftarrow V \cup V'$ 
19    $EST[q] \leftarrow e$ 
20    $Vu \leftarrow Vu \cup \{q\}$ 
21 until  $Vu = V$ 

22  $P \leftarrow \{v \in V | v \text{ dans la composante fortement}$ 
    $\text{connexe puits de } (V, E)\}$ 
23  $v \leftarrow \min\{EST[q] | q \in P\}$ 
24 decide( $v$ )
25 halt

```

Figure 1: Consensus avec Σ

Si $p \neq z$, Si z est dans V_p , pour le processus p , la deuxième itération ne pourra se terminer que si z appartient à l'ensemble Vu de p . Pour que z soit inclus à cet ensemble il faut que p reçoive un message (2, $z, W_z, *$). p ajoutera alors W_z à son ensemble V_p et tous les arcs issu de z et à destination d'un processus de W_z à E_p . Ensuite E_p et V_p ne font que croître. De plus aucun arc issu de z ne sera ajouté ensuite dans la deuxième itération. \square

LEMME 4. *Pour tous les processus, les graphes obtenus par l'algorithme de la Figure 1 possèdent la même composante fortement connexe puits.*

PREUVE. Soient p et q deux processus, par le lemme 2, G_p (resp. G_q) a une composante connexe puits P_p (resp. P_q). On va montrer que $P_p \subseteq P_q$.

Soient x un processus élément de P_p et y un processus élément de P_q . W_x et W_y contiennent une sortie de Σ .

Par la propriété d'intersection de Σ , W_x et W_y ont un élément commun z . D'après le lemme 3, x étant dans V_p , z est dans V_p . De même, z appartient à V_q . D'après le lemme 3, quand l'arc (x, z) appartient à E_p et l'arc (y, z) appartient à E_q .

Comme x appartient à P_p , et qu'il n'y a aucune arc sortant de P_p , z appartient à P_p . Il y a donc un chemin de z à x (dans G_p). Soient $x_1 = z, x_2, \dots, x_k = x$ ce chemin. D'après le lemme 3, si x_i est dans V_p alors x_{i+1} est dans W_{x_i} . On a alors d'après le lemme 3, x_i dans V_q et (x_i, x_{i+1}) dans E_q . Comme il y a un arc de y à z dans G_q , il y a un chemin de y

à x dans G_q . Comme y appartient à P_q et qu'il n'y a aucun arc sortant de P_q , x appartient à P_q . \square

THÉORÈME 5. *L'algorithme de la Figure 1 implémente un consensus.*

PREUVE.

Accord: Tous les processus qui exécutent la ligne 22 peuvent extraire une composante fortement connexe puits par le Lemme 2. Cette composante est identique par le Lemme 4. Chaque processus de la composante fortement connexe ayant envoyé la valeur qu'il propose au consensus dans les messages de type 2 qui a permis de constituer le graphe, tous les processus décident la même valeur.

Intégrité: La décision se fait sur la valeur proposée par un processus.

Terminaison: La terminaison de la boucle *repeat until* des lignes 7 à 12 est assurée par la propriétés de complétude de Σ . Le système que l'on considère ici assure que tous les processus dont un message a été reçu par un autre processus dans cette boucle enverront un deuxième message. Ce qui assure la terminaison de la deuxième boucle *repeat until* des lignes 14 à 21. \square

4. TRAVAUX RELIÉS

Dans [5], les auteurs présentent plusieurs algorithmes d'élection de leader dans des systèmes partiellement synchrones sans la connaissance complète du nombre de participants et des défaillances de type "crash". Ils prouvent que sous certaines hypothèses de synchronie des liens de communication cette élection de leader est possible sans la connaissance du nombre de participants. L'élection de leader permet de résoudre le problème du consensus avec une *majorité* de processus corrects (il faut de plus que les processus connaissent ce nombre). A partir de ces résultats et de ceux présentés ici on peut définir les conditions qui permettent de résoudre le consensus dans des modèles partiellement synchrones sans la connaissance des participants: d'une part des conditions de synchronie permettant d'élire de façon ultime un leader et d'autre part des conditions permettant de réaliser Σ . On retrouve ainsi le résultat montrant que le plus faible détecteur de défaillances pour résoudre le consensus est $\Sigma \times \Omega$ (Σ assure les propriétés de quorum et Ω permet l'élection de leader).

Notons qu'un problème important dans le contexte des réseaux dans lequel l'ensemble de processus n'est pas connu est celui de la découverte des divers sites. Un première approche dans ce sens est [1, 7] où les auteurs présentent des conditions permettant de résoudre le consensus dans ce contexte.

5. CONCLUSION ET PERSPECTIVES.

Sans la connaissance de l'ensemble des processus, le consensus n'est pas possible à résoudre même avec des restrictions très fortes sur le modèle de défaillances. On a vu que l'introduction du détecteur de défaillances Σ est un moyen de contourner les résultats d'impossibilité. Cependant ce résultat reste théorique dans la mesure où il détermine quelle

information sur les défaillances est nécessaire, mais le problème à résoudre alors est de déterminer des conditions sur le système qui permettent d'implémenter un tel détecteur de défaillances. Nous avons vu que c'est possible avec une estimation du nombre de processus vivants. Dans le cas où on ne dispose pas de cette estimation, il faut, sans doute, faire des hypothèses supplémentaires de synchronie sur le système. Déterminer quelles hypothèses faire est un problème ouvert.

6. REFERENCES

- [1] David Cavin, Yoav Sasson, and André Schiper. Consensus with unknown participants or fundamental self-organization. In Ioanis Nikolaidis, Michel Barbeau, and Evangelos Kranakis, editors, *ADHOC-NOW*, volume 3158 of *Lecture Notes in Computer Science*, pages 135–148. Springer, 2004.
- [2] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.
- [3] Carole Delporte-Gallet, Hugues Fauconnier, and Rachid Guerraoui. Shared memory vs. message passing. Technical Report IC/2003/77, EPFL, December 2003. Available at <http://icwww.epfl.ch/publications/>.
- [4] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, Vassos Hadzilacos, Petr Kouznetsov, and Sam Toueg. The weakest failure detectors to solve certain fundamental problems in distributed computing. In Soma Chaudhuri and Shay Kutten, editors, *PODC*, pages 338–346. ACM, 2004.
- [5] Antonio Fernández, Ernesto Jiménez, and Michel Raynal. Eventual leader election with weak assumptions on initial knowledge, communication reliability, and synchrony. In *DSN*, pages 166–178. IEEE Computer Society, 2006.
- [6] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.
- [7] Fabíola Greve and Sébastien Tixeuil. Knowledge connectivity vs. synchrony requirements for fault-tolerant agreement in unknown networks. In *DSN*, pages 82–91. IEEE Computer Society, 2007.
- [8] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.