Tolerance aux fautes et systemes repartis : concepts et mecanismes

Gerard Le Lann, Pascale Minet, David Powell

Collection

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Tolérance aux fautes et systèmes répartis : Concepts et mécanismes

Gérard LE LANN
Pascale MINET
David POWELL

N° 2108
Novembre 1993

PROGRAMME 1

Architectures parallèles,
bases de données,
réseaux et systèmes distribués

TOLERANCE AUX FAUTES ET SYSTEMES REPARTIS : Concepts et Mécanismes[1]

Gérard Le Lann, Pascale Minet, David Powell

Gerard.Le_Lann@inria.fr, Pascale.Minet@inria.fr
INRIA, BP 105, Rocquencourt, 78153 Le Chesnay Cedex

David.Powell@laas.fr
LAAS-CNRS, 7 Avenue du Colonel Roche, 31077 Toulouse Cedex

Résumé :

L'exigence de tolérance aux fautes est incontournable avec les systèmes répartis. D'une part, la tolérance aux fautes est un besoin induit par la multiplicité des ressources et de nombreux travaux sur les systèmes répartis ont pour but de garantir que la sûreté de fonctionnement de ces systèmes n'est pas dégradée par la répartition.

Par ailleurs, la tolérance aux fautes peut être en elle-même un facteur motivant la répartition. En effet, la tolérance aux fautes ne peut être assurée sans redondance et la répartition des traitements et des données sur des processeurs différents permet de structurer et gérer cette redondance.

Dans ce rapport, nous présentons les concepts et mécanismes utilisés dans les systèmes répartis tolérants aux fautes. Une application répartie peut être vue soit comme un ensemble de processus interagissant par échange de messages, soit comme un ensemble de transactions manipulant des données réparties. Les techniques connues de tolérance aux fautes sont fournies pour ces deux schémas de traitement. Nous introduisons ensuite les concepts de diffusion et de consensus réparti tolérant aux fautes.

1. Ce travail a été réalisé dans le cadre du groupe de travail "Informatique Tolérante aux Fautes" de l'OFTA

FAULT TOLERANCE AND DISTRIBUTED SYSTEMS
Concepts and Mechanisms[1]

Gérard Le Lann, Pascale Minet, David Powell

Gerard.Le_Lann@inria.fr, Pascale.Minet@inria.fr
INRIA, BP 105, Rocquencourt, 78153 Le Chesnay Cedex

David.Powell@laas.fr
LAAS-CNRS, 7 Avenue du Colonel Roche, 31077 Toulouse Cedex

Abstract:

Fault-tolerance is an unavoidable requirement in distributed systems. First, multiple resources imply multiple potential causes of failure so much research on distributed systems has aimed to ensure that dependability is not degraded by distribution. Second, fault tolerance can in itself be a motivating factor for distribution. Indeed, fault tolerance cannot be ensured without redundancy, and the distribution of processing and data on different processors provides an approach for structuring and managing this redundancy.

In this report, we present the concepts and mechanisms used in fault tolerant distributed systems. A distributed application can be considered either as a set of processes exchanging messages or as a set of transactions acting on distributed data items. The known techniques of fault-tolerance are given for both computational models. We then introduce the concepts of fault tolerant group commucation and distributed consensus.

1. This work was carried out in the OFTA working group "Informatique Tolérante aux Fautes"

1. Introduction

Un système réparti est défini comme un ensemble de noeuds de traitement ou processeurs (avec mémoire incorporée) interconnectés par un réseau de communication et communiquant uniquement par messages. Lamport, cité dans [Mullender 1989], en donne une autre définition : "un système réparti est un système qui vous empêche de travailler lorsqu'une machine dont vous n'avez jamais entendu parlé s'écroule" ! De façon plus sérieuse mais néanmoins similaire, l'incertitude introduite (entre autre) par l'existence de délais variables et par la non-sûreté de fonctionnement des processeurs de traitement et des moyens de communication a pour conséquence l'impossibilité d'observer l'état global d'un système réparti [Lamport 1978] [Le Lann 1977], ce qui est la caractéristique principale permettant de distinguer la classe des systèmes "répartis" de celle des systèmes "parallèles" [Fischer 1990]. La boutade de Lamport et la distinction de Fischer entre systèmes "répartis ou parallèles" soulignent que la sûreté de fonctionnement est un souci inhérent aux systèmes répartis. En effet, la tolérance aux fautes est un besoin induit par la répartition et de nombreux travaux sur les systèmes répartis prennent en compte la possibilité de défaillances afin d'éviter que la sûreté de fonctionnement ne soit dégradée du fait de la répartition.

Par ailleurs, les systèmes informatiques dits "tolérants aux fautes", dont l'objectif principal est d'assurer une très forte sûreté de fonctionnement, s'appuient souvent sur des architectures réparties. En effet, la tolérance aux fautes ne peut être assurée sans redondance et la répartition des traitements et des données sur des processeurs différents est une solution intéressante pour structurer et gérer cette redondance. Par conséquence, la tolérance aux fautes est un facteur motivant la répartition.

Ces liens étroits entre tolérance et répartition sont à l'origine de nombreux travaux visant à définir les concepts et les mécanismes de tolérance aux fautes dans les systèmes de processeurs répartis communiquant seulement par échange de messages. La tolérance aux fautes dans de tels systèmes est mise en œuvre principalement par logiciel, ce qui permet d'espérer une certaine pérennité des moyens de tolérance vis-à-vis des évolutions technologiques du matériel. Par ailleurs, si les processeurs sont géographiquement séparés, la probabilité qu'un désastre affecte plusieurs processeurs est diminuée et la tolérance aux désastres peut donc être fournie en même temps que la tolérance aux fautes internes aux processeurs. La tolérance aux fautes consiste, bien sûr, à assurer que l'application continue à fournir un service spécifié malgré la présence de fautes. Le fait qu'il s'agit d'un système réparti (plutôt qu'une machine tolérante aux fautes fortement couplée) ne change pas cet objectif. Il a cependant les conséquences suivantes :

- le traitement d'erreurs et de fautes doit s'effectuer principalement par messages ;
- la granularité naturelle d'application des techniques de recouvrement, de masquage et de reconfiguration est celle des processus ou serveurs (modèles processus/message et client/serveur) ou bien des données (modèle transaction/ données) ;
- malgré les fautes et la concomitance des exécutions, l'état global du système doit être maintenu cohérent, bien que non directement observable ni a fortiori manipulable.

2. Concepts

2.1. Modèles conceptuels temporels

La conception d'un système réparti est basée sur un modèle conceptuel temporel, qui spécifie les hypothèses temporelles de départ. Il existe trois modèles conceptuels en ce qui concerne les délais : le modèle Délais Bornés et Connus (DBC), le modèle Délais Bornés mais Inconnus (DBI) et le modèle Délais Non Bornés (DNB). Ces modèles se différencient par rapport aux hypothèses de connaissance à la conception (d'une solution) concernant les bornes minimales et maximales sur les délais de communication et de traitement :

- **Modèle DBC** : les bornes existent et leurs valeurs sont connues a priori.
- **Modèle DBI** : les bornes existent, mais leurs valeurs sont inconnues.
- **Modèle DNB** : les bornes n'existent pas.

Pour un problème donné, le choix du modèle est extrêmement important car il détermine les solutions. Ainsi tout problème de consensus a des solutions probabilistes ou aléatoires (randomized) dans les 3 modèles, des solutions déterministes seulement dans les modèles DBC et DBI. A quelques exceptions près, les solutions présentées dans ce chapitre correspondent à un modèle DBC.

2.2. Entraves à la sûreté de fonctionnement

La prémisse de base de la tolérance aux fautes dans les systèmes répartis est l'indépendance des défaillances des différents processeurs du système. En ce qui concerne les fautes à l'origine des défaillances, cette prémisse semble satisfaisante en ce qui concerne les fautes physiques internes aux processeurs. On peut aussi argumenter de sa justification pour d'autres types de fautes. Par exemple, la répartition géographique et un couplage faible entre processeurs de traitement permettent aussi de supposer que les fautes accidentelles externes et certaines fautes de conception du logiciel se manifesteront de façon indépendante sur différents processeurs. Par la suite, nous ne faisons pas de distinction entre l'orgine des défaillances des processeurs — nous supposons seulement que ces défaillances ont lieu de façon indépendante.

La recherche sur la tolérance aux fautes dans les systèmes répartis a donné lieu à une certaine formalisation de la façon par laquelle les fautes se manifestent sur le comportement d'un processus ou d'un processeur de traitement [Powell 1992]. Les modes de défaillances les plus couramment étudiés sont (par ordre croissant de sévérité) [Cristian, Aghali, Strong, Dolev 1985] : **les défaillances par arrêt, les défaillances par omission, les défaillances temporelles et les défaillances arbitraires.** Il est clair que les défaillances temporelles n'ont aucun sens dans un modèle DNB.

Il convient de considérer aussi certaines abstractions simplificatrices en ce qui concerne la structure interne des processeurs. La notion de mémoire stable est une abstraction bien connue qui permet d'assurer la survie des données stockées au-delà d'une défaillance d'un processeur [Lampson 1981]. Plus généralement, on peut distinguer, au niveau de la mémoire interne d'un processeur, trois types de mémoire

[Spector 1989] :

- **la mémoire volatile** dont le contenu est perdu si le processeur défaille. Lors de son recouvrement, un processeur disposant seulement d'une telle mémoire est amnésique ;
- **la mémoire non-volatile** dont le contenu peut survivre à la défaillance du processeur. Le contenu de cette mémoire peut avoir été contaminé lors de la défaillance du processeur. Le recouvrement du processeur disposant d'une telle mémoire doit comprendre l'évaluation d'une éventuelle contamination.
- **la mémoire stable** dont le contenu survit toujours à la défaillance du processeur. Lors de son recouvrement, un processeur disposant d'une mémoire stable peut utiliser sans crainte les informations contenues dans cette mémoire.

Une autre abstraction au niveau d'un processeur de traitement est celle d'un **contrôleur de communication à silence sur défaillance** [Powell et al 1988] ; cette abstraction permet de concevoir des protocoles de communication (et de traitement des erreurs) sous l'hypothèse de défaillances par arrêt uniquement, sans pour autant limiter les modes de défaillance admis au niveau des processeurs de traitement.

2.3. Schémas de traitement réparti

La tolérance aux fautes dans les systèmes répartis peut être abordée selon deux approches apparamment différentes selon que l'on considère un schéma de traitement "opérations distribuées sur des données réparties" ou un schéma "processus communicants".

Dans le premier cas, une application répartie consiste en l'exécution d'opérations réparties qui ont pour but de faire passer les données réparties d'un état cohérent à un autre état cohérent, les contraintes de cohérence étant définies selon l'application. Dans cette approche, une opération distribuée peut être abandonnée pour des raisons purement logiques, par exemple, si la valeur d'une donnée répartie (solde de compte bancaire insuffisant) ne permet pas de mener l'opération (transfert de fonds depuis ce compte) à sa fin. Il est alors nécessaire de sauvegarder (ou de pouvoir reconsituer) l'état des données réparties tel qu'il existait au début de l'exécution de l'opération. Les points de reprise ainsi constitués, et l'organisation du traitement en opérations atomiques, fournissent aussi un cadre idéal pour la mise en œuvre d'une politique de tolérance aux fautes répartie par détection d'erreurs et recouvrement arrière [Laprie 1992]. Cette approche est développée au paragraphe 4.

Dans le deuxième cas, une application répartie est vue comme un ensemble de "processus" qui interagissent par échange de messages [Raynal 1991]. On fait abstraction dans ce cas de la sémantique des messages échangés. Bien que l'état interne d'un processus puisse être vu en quelque sorte comme une "donnée répartie", le schéma de traitement "processus communicants" ne précise pas l'effet des messages reçus sur cet état interne — on suppose simplement que cet état est éventuellement modifié par tout message. Dans ce cas, la tolérance aux fautes répartie est abordée soit au niveau de chaque processus pris individuellement (par exemple, par le biais de processus répliqués) soit au niveau d'un ensemble délimité de processus communicants (par exemple, par le biais de conversations). Cette approche est développée au paragraphe suivant.

3. Processus communicants

Les techniques réparties de tolérance aux fautes avec un schéma de traitement par processus communicants peuvent être classées en deux catégories :

- les techniques s'appuyant sur l'existence d'une mémoire stable au niveau de chaque processeur de traitement ou accessible par plusieurs processeurs de traitement ;
- les techniques à base de réplication de processus sur des processeurs différents.

3.1. Tolérance aux fautes à base de mémoire stable

L'abstraction d'une mémoire stable permet, comme dans l'approche transactionnelle développée dans le paragraphe suivant, de mettre en œuvre des techniques de tolérance aux fautes basées sur la détection d'erreur et le recouvrement arrière. Les techniques de tolérance aux fautes à base de mémoires stables locales sont limitées car les informations sauvegardées restent indisponibles tant que le processeur défaillant en question n'effectue pas un recouvrement ou est réparé. Une alternative à la notion de mémoire stable locale est l'utilisation d'une (ou de plusieurs) mémoires stables accessibles depuis plusieurs machines. Dans le cadre des architectures réparties considérées ici (ensemble de processeurs interconnectés par un réseau de communication et communiquant uniquement par messages, cf §1.), une telle mémoire stable répartie pourrait prendre la forme d'un ou plusieurs serveurs de mémorisation stable.

La tolérance aux fautes par sauvegarde périodique de points de reprise sur mémoire stable s'appuie presque toujours sur l'hypothèse que les processeurs défaillent seulement par arrêt : en particulier toute information sauvegardée sur mémoire stable ou envoyée vers les autres processeurs est supposée correcte. Une exception à cette règle est l'approche développée dans [Schlichting, Schneider 1983], [Schneider 1984] où l'abstraction d'un processeur à arrêt sur défaillance et mémoire stable globalement accessible est obtenue au moyen d'un protocole de coordination entre plusieurs processeurs élémentaires admettant des défaillances arbitraires.

La tolérance aux fautes par sauvegarde périodique de points de reprise sur mémoire stable est très aisément mise en œuvre dans le cas d'une application séquentielle (et donc non-répartie). Cependant, la répartition d'une application sur plusieurs processeurs ajoute une dimension supplémentaire au problème car il faut éviter que la reprise de l'exécution par un processus introduise des incohérences dans les exécutions effectuées par les autres processus. Le problème de la tolérance aux fautes par recouvrement arrière consiste à assurer que, malgré la reprise d'un processus, l'état global du système de processus reste cohérent [Chandy, Lamport 1985]. Pour ce faire, il existe trois catégories d'approches :

1) Les processus enregistrent des points de reprise de façon asynchrone et indépendamment les uns des autres. Lors de la défaillance d'un processus, les processus doivent rechercher un ensemble de points de reprise parmi ceux sauvegardés qui représente un état global cohérent. Pour éviter l'effet domino [Randell 1975], on doit avoir recours à l'enregistrement des messages reçus depuis le dernier point de reprise. Ces messages sont alors "rejoués" pendant la ré-exécution (on est alors obligé de

supposer que les processus sont déterministes) (voir, par exemple, [Johnson, Zwaenepoel 1990], [Strom, Yemini 1985], [Le Lann 1981]).

2) L'enregistrement des points de reprise est pré-programmé afin de générer un ensemble de points de reprise correspondant à un état global cohérent. Cela peut se faire soit en prenant un point de reprise local avant chaque envoi de message, ce qui peut être très pénalisant en performance, soit au moyen de conversations [Randell 1975]. Il a été démontré qu'il existe une dualité entre l'approche "processus/conversations" et l'approche "données/transactions" [Shrivastava, Mancini, Randell 1987], développée au paragraphe 4, pour ce qui concerne le recouvrement d'erreur (mais pas la concomitance).

3) On assure une coordination dynamique entre les actions d'enregistrement de points de reprise de telle façon que l'ensemble des points de reprise représente un état global cohérent. Cette approche évite le problème de l'effet domino de façon transparente au programmeur, même si les processus ne sont pas déterministes, au prix de la pénalité de performance imposée par l'exécution du protocole de coordination [Koo, Toueg 1987], [Leu, Bhargava 1989].

3.2. Tolérance aux fautes basée sur la réplication de processus

Pour certaines applications, les techniques de tolérance à base de mémoire stables présentent des inconvénients rédhibitoires. Dans le cas de mémoire stables locales aux processeurs, il est bien évident que toute défaillance d'un processeur rend inaccessible les points de reprises qui y sont stockés, ceux-ci ne peuvent pas être utilisés tant que le processeur est défaillant. En conséquence, les techniques de tolérance à base de mémoires stables locales ne permettent pas un accroissement de la disponibilité d'une application par rapport à celle qui résulterait de l'exécution centralisée du même traitement sur un seul processeur. On peut pallier cet inconvénient en recourant à un serveur de mémorisation stable. Cependant, il est bien évident qu'un tel serveur constitue un point dur vis-à-vis de la tolérance aux fautes (il devra donc être lui aussi tolérant aux fautes) et peut devenir un goulot d'étranglement.

Une autre approche consiste alors en la création des copies multiples des processus sur des processeurs différents. Par exemple, dans le projet Delta-4 [Powell 1991], trois techniques de réplication ont été approfondies et mises en œuvre :

- **la réplication active** est une technique par laquelle toutes les répliques traitent tous les messages d'entrée (diffusés à l'ensemble du groupe) de façon concomitante afin de garder leurs états internes étroitement synchronisés. Cette technique permet en particulier de mettre en œuvre un vote sur les sorties afin de se prémunir contre les défaillances arbitraires [Chérèque et al 1992].
- **la réplication passive** est une technique par laquelle seule une des répliques (la réplique primaire) traite les messages d'entrée et fournit les messages de sortie. En l'absence de fautes, les autres répliques (les répliques de secours ou répliques secondaires) ne traitent pas les messages de sortie ; leur état interne est cependant régulièrement mis à jour au moyen de points de reprise envoyés par la réplique primaire. Cette technique relève de la détection et du recouvrement arrière comme les techniques à base de mémoire stable décrites au paragraphe précédent — les

répliques passives servent en quelque sorte de "mémoires stables" pour les points de reprise de la réplique primaire.

• **la réplication semi-active** est une technique hybride entre la réplication active et la réplication passive ; les messages d'entrée sont difusés à toutes les répliques mais une seule des répliques (la réplique leader) traite tous les messages d'entrée et fournit les messages de sortie. En l'absence de fautes, les autres répliques (les répliques suiveuses) ne produisent pas de messages de sortie ; leur état interne est mis à jour soit par traitement direct des messages d'entrée, soit au moyen de "notifications" de la réplique leader.

Ces différentes techniques de réplication fournissent un moyen de traiter les erreurs et de masquer ainsi le fait que des répliques ont défailli. Cependant, afin de restaurer le niveau de redondance des processus et d'être ainsi capables de tolérer d'autres défaillances, ces techniques doivent être complétées par un traitement de fautes, qui consiste, entre autres, en la ré-allocation et l'initialisation de nouvelles répliques.

4. Opérations distribuées sur des données réparties

Dans ce paragraphe nous nous intéressons aux opérations distribuées préservant la cohérence des données. Nous définissons tout d'abord la cohérence des données, puis les propriétés ACID des opérations distribuées. Nous examinons ensuite comment ces propriétés peuvent être maintenues.

4.1. Cohérence des données

La **cohérence interne** d'un système d'information (représenté par l'ensemble des objets accessibles) est exprimée sous forme d'invariants ou contraintes de cohérence. Une contrainte de cohérence peut concerner un seul objet (par exemple l'âge d'une personne est comprise entre 0 et 130 ans) ou plusieurs objets (lors d'une opération de transfert bancaire de X vers Y le montant débité sur le compte X est égal au montant crédité sur le compte Y).

Lorsqu'un objet existe en multiples exemplaires, cet objet est dit être maintenu en copies multiples. Remarquons que l'existence de copies multiples peut résulter soit des impératifs de la tolérance aux fautes, soit d'un souci d'amélioration des performances en lecture (accès à une copie locale). La **cohérence mutuelle** désigne la cohérence entre les différentes copies de ce même objet.

4.2. Opérations distribuées atomiques

Une opération distribuée est constituée d'opérations locales, chaque opération locale s'exécutant sur un seul processeur. Nous nous intéressons plus particulièrement aux **opérations distribuées atomiques ou transactions**, car elles préservent la cohérence des données. Par définition les opérations distribuées atomiques ou transactions satisfont les propriétés **ACID** [ISO/IEC 1992] : Atomicité, Cohérence, Isolation et Durabilité. Ces propriétés sont définies comme suit :

• **Atomicité** : les effets d'une transaction ne sont rendus visibles que si toutes ses opérations peuvent être réalisées, sinon aucun effet n'est produit par la transaction.

• **Cohérence** : les effets d'une transaction doivent satisfaire les contraintes de cohérence du système d'information.

• **Isolation** : les effets d'une transaction sont identiques à ceux qu'elle produirait si elle s'exécutait seule dans le système.

• **Durabilité** : les effets d'une transaction globalement terminée ne peuvent pas être détruits ultérieurement par une quelconque défaillance.

Remarques : Une transaction, prise individuellement, préserve la cohérence des objets. La propriété de Durabilité implique que tout processeur participant à l'exécution d'une transaction dispose d'une mémoire stable.

4.3. Transactions, concomitance et défaillances

Le modèle transaction/données présente l'avantage de supporter naturellement le concept d'architectures ou d'applications évolutives. Chaque transaction est implantée correctement et séparément. Le développement de nouvelles transactions ne conduit pas à une remise en cause des transactions existantes. De même le changement d'architecture ne remet pas en cause la conception préalable des transactions. La mise au point de l'application est facilitée, puisqu'il suffit de mettre au point chaque transaction séparément.

Bien que chaque transaction prise individuellement préserve la cohérence des objets, cette cohérence des objets peut être détruite par la concomitance[1] et par les défaillances. **Le maintien des propriétés ACID des transactions durant leur exécution (en présence de traitements concomitants et de défaillances) est à la charge d'algorithmes en ligne,** car il est en général impossible de prédire les conditions dans lesquelles les propriétés ACID peuvent être violées.

Le modèle transactionnel est employé depuis longtemps dans de nombreusess applications en-ligne (ex: réservations, marchés financiers...). Par contre, ce modèle est encore peu utilisé dans le monde du temps réel. Des travaux sont en cours pour établir les propriétés temporelles découlant de solutions combinées permettant de résoudre les trois problèmes de concomitance, de consensus réparti fiable et d'ordonnancement par attributs temporels [Le Lann 1992].

4.3.1. Maintien des propriétés ACID en présence d'exécutions concomitantes

Le maintien des propriétés ACID en présence d'exécutions concomitantes est à la charge d'algorithmes de contrôle de concomitance. Il a été démontré qu'une exécution sérialisable des transactions (équivalente au niveau de ses effets sur les objets à une exécution en série des transactions) permet d'obtenir cette propriété. Le rôle du contrôle de concomitance consiste donc à n'autoriser que des exécutions sérialisables. Il existe différents types d'algorithmes de contrôle de concomitance [Bernstein, Hadzilacos, Goodman 1987] : verrouillage, estampille, certification.

Ainsi un contrôle de concomitance basé sur un verrouillage en deux phases, avec prévention des interblocages par retour arrière de la transaction la plus jeune, permet

1. le terme concomitance est préféré au terme concurrence car il exprime mieux la simultanéité reflètée dans le terme anglais concurrency.

d'offrir en plus des propriétés ACID, la garantie de terminaison certaine en l'absence de défaillance, de toute transaction soumise au système.

La fiabilisation de l'algorithme de contrôle de concomitance (comportement de l'algorithme en présence de défaillance) et le maintien des propriétés ACID de la transaction en présence de défaillance font l'objet du paragraphe suivant.

4.3.2. Maintien des propriétés ACID en présence de défaillances

La validation en deux phases d'une opération distribuée est la solution qui permet de maintenir les propriétés ACID de cette opération en présence de défaillances. Lorsque la transaction a terminé son exécution, le protocole de validation à deux phases doit s'assurer que tous les processeurs impliqués dans l'exécution de cette transaction sont d'accord pour rendre permanents les effets de cette transaction, c'est-à-dire **valider** la transaction. Si l'unanimité des processeurs impliqués n'est pas obtenue, la transaction est annulée. La non-unanimité peut provenir du désaccord de l'un des processeurs impliqués ou de l'impossibilité de communiquer avec l'un des processeurs impliqués (ex. : processeur isolé ou défaillant).

Le protocole de validation à 2 phases est invoqué par un processeur particulier : **le coordinateur**. Ce processeur est généralement celui qui a soumis la transaction. Les processeurs impliqués dans l'exécution de la transaction sont appelés **participants.** Comme son nom l'indique, le protocole se déroule en deux phases.

Dans la première phase, le coordinateur, averti de la fin d'exécution de la transaction, diffuse à tous les participants un message leur demandant s'ils sont prêts à valider. Sur réception de ce message, un participant prend les mesures nécessaires pour être en mesure de rendre permanents les effets de la transaction en cours de validation (écriture en mémoire stable). Puis il répond au coordinateur en donnant son accord.

Dans la deuxième phase, le coordinateur qui a reçu l'accord de tous les participants, diffuse l'ordre de valider. Sur réception de ce message, un participant rend permanents les effets de la transaction qui est dorénavant validée, c'est-à-dire écrit en mémoire stable l'état validé de la transaction, reporte les valeurs après mise-à-jour et libère les verrous si le contrôle de concomitance est basé sur un verrouillage en deux phases. Puis il acquitte la validation au coordinateur. Si par contre le coordinateur est dans l'impossibilité de recevoir tous les accords, le coordinateur diffuse l'ordre d'annuler la transaction. Cet ordre est acquitté par les participants.

Les premières descriptions non publiées de ce protocole remontent à 1975 [Lindsay et al 1979]. La première description d'une version répartie tolérant les défaillances du coordinateur et des participants remonte à 1981 [Le Lann 1981].

Tolérance aux fautes

La défaillance d'un participant se traduit par la non-réception de sa réponse dans le délai imparti. La défaillance du coordinateur se traduit par la non-réception de l'ordre de validation ou annulation dans le délai imparti, ceci sous-entend un modèle TBC (cf §2.1.) et des défaillances par arrêt (cf §2.2.).

Si un participant défaille en première phase (c'est-à-dire, son accord n'est pas parvenu

au coordinateur), la transaction est annulée. Si un participant défaille après avoir fait parvenir son accord au coordinateur, lorsqu'il sera réinséré dans le système, il consultera sa mémoire stable, puis interrogera le coordinateur de cette transaction qui lui répétera l'ordre de valider ou d'annuler.

Si le coordinateur défaille en deuxième phase, les participants se consultent. Si l'un d'entre eux a reçu l'ordre de valider, alors la transaction est validée. De même si l'un d'entre eux a reçu l'ordre d'annuler, alors la transaction est annulée. Dans tout autre cas et si tous les participants sont d'accord pour valider, alors la transaction est validée.

Si le coordinateur a défailli ainsi qu'un participant et aucun des participants présents n'a reçu l'ordre de valider ou d'annuler, il y a indécidabilité jusqu'à la réinsertion du participant défaillant. En effet quelque soit la décision prise par les participants présents, celle-ci peut être contredite par le participant défaillant : ce qui violerait l'atomicité de la transaction.

4.4. Réplication des données

La principale raison de l'utilisation de données en copies multiples réside dans la garantie d'une meilleure **disponibilité** de ces données. L'existence de copies multiples doit être rendue transparente à l'utilisateur par **un algorithme de gestion des copies multiples**. Cet algorithme couplé à un contrôle de concomitance tel que décrit précédemment doit garantir que toute exécution autorisée est équivalente à une exécution sérielle des transactions sur une seule copie (**"one copy serialisability"**). Le coût induit par cet algorithme de contrôle de concomitance (ex.: écriture de w copies au lieu d'une seule) doit être faible.

On distingue deux classes d'algorithmes de gestion des copies multiples selon qu'ils tolèrent ou non le partitionnement du système réparti.

4.4.1. Algorithmes ne tolérant pas le partitionnement

Ces algorithmes adoptent l'hypothèse suivante : **toute copie localement accessible est accessible à distance**. Autrement dit, les processeurs sont à arrêt sur défaillance et la communication est sans défaillance.

L'**algorithme des copies disponibles** [Bernstein, Hadzilacos, Goodman 1987] utilise la technique dite ROWA ("Read One, Write All") qui consiste à lire une copie (généralement la copie locale) et à écrire sur toutes les copies (ici toutes les copies disponibles). Cet algorithme peut être utilisé avec un verrouillage strict en deux phases, ou avec des estampilles comme dans SDD-1 [Bernstein, Shipman, Rothnie 1980]. Si un rejet est reçu (incompatibilité d'accès) ou si une écriture n'est acquittée par aucune copie, alors la transaction est annulée. Dans tout autre cas la transaction est validable. La validation comprend une phase supplémentaire par rapport au protocole de validation en deux phases classique. Cette phase supplémentaire vise à s'assurer que toutes les copies accessibles ont connaissance de l'écriture. Pour **remettre à niveau** une copie, une transaction spécifique est exécutée : elle lit une copie et écrit ce qu'elle a lu sur la copie à remettre à niveau. Si l'on veut pouvoir créer/détruire dynamiquement des copies ou en changer l'emplacement, il faut disposer d'un **répertoire** des copies disponibles : ce répertoire existe lui aussi en copies multiples.

On peut également citer dans cette classe l'**algorithme de la copie privilégiée**

[Alsberg, Day 1976], [Stonebraker 1979]. Les écritures sont effectuées sur la copie privilégiée. La copie privilégiée est chargée de reporter toutes les écritures sur les autres copies. Ce report est généralement effectué en tâche de fond. Sur défaillance de la copie privilégiée, une autre copie devient copie privilégiée. Afin d'éviter qu'au cours de ce basculement, des transactions ne soient perdues (écritures associées non reportées sur la nouvelle copie privilégiée), la copie privilégiée doit attendre l'acquittement de l'écriture sur k copies. L'algorithme est alors ***k*-résilient**. Remarquons qu'on peut établir un parallèle entre cet algorithme et la réplication semi-active (cf §3.2.).

4.4.2. Algorithmes tolérant le partitionnement

Ces algorithmes supposent que les processeurs sont sujets à des défaillances par arrêt ou par omission et que la communication est également sujette à des défaillances par omission. **Un groupe d'une ou plusieurs copies peut donc être isolé des autres copies du système.**

Ces algorithmes associent à chaque copie un poids (entier positif) et un numéro de version. Seul le groupe totalisant un certain **quorum** est autorisé à travailler. On distingue trois sous-classes :

- algorithmes du quorum de copies,
- algorithmes de la partition virtuelle,
- algorithmes à quorum dynamique.

a) Algorithmes du quorum de copies

Chaque opération de lecture (resp. d'écriture) nécessite un certain quorum [Gifford 1979], [Herlihy 1986] dit **quorum de lecture** QL (resp. **quorum d'écriture** QE). Les quorums doivent être tels que deux opérations conflictuelles (L/E ou E/E) ne peuvent s'exécuter simultanément. Une lecture se traduit donc par la lecture de QL copies, la valeur retournée est celle fournie par la copie de plus grand numéro de version. Une écriture se traduit par la lecture de QE copies et l'écriture sur ces QE copies de la nouvelle version réalisée à partir de la copie lue de plus grand numéro de version v. Cette nouvelle version porte le numéro $v+1$.

Une variante de cet algorithme (algorithme **des écritures manquantes** [Eager, Sevcik1983]) a été introduite afin de limiter le surcoût induit par les quorums. Elle consiste à n'utiliser les quorums qu'en **mode défaillance** (défaillance de processeur ou communication). En **mode normal** c'est l'algorithme des copies disponibles qui est utilisé. Lorsqu'une transaction détecte une défaillance (non-réception d'un acquittement d'écriture) elle doit se réexécuter en mode défaillance et propager cette information à toutes les transactions dépendantes au moyen d'une liste L maintenue sur chaque copie accédée. Une transaction Ti dépend d'une transaction Tj si Ti a lu ou écrit un objet après que Tj ait écrit cet objet. La **remise à niveau** d'une copie est faite par une transaction spécifique qui lit QL copies et écrit sur toutes ces copies la version la plus à jour, puis met à jour la liste L sur toutes les copies.

Les algorithmes dits **à consensus majoritaire** [Thomas 1979] appartiennent à cette sous-classe.

b) Algorithmes de la partition virtuelle

A la différence des algorithmes basés sur un quorum, où le quorum de copies nécessaire à la réalisation d'une opération est fixé une fois pour toutes, les algorithmes de la partition virtuelle adoptent une approche dynamique : le nombre de copies nécessaire à la réalisation d'une écriture est égal au nombre de processeurs présents. L'algorithme de la **partition virtuelle** est décrit dans [El Abbadi, Skeen, Cristian 1985]. Chaque processeur possesseur d'une copie maintient sa **vue** qui est la liste des processeurs possesseurs de copies avec qui il communique. Une lecture se traduit par la lecture d'une copie quelconque de la vue de la transaction (c'est-à-dire la vue du processeur initiateur de la transaction) ; une écriture se traduit par l'écriture de toutes les copies de la vue. Si en cours d'exécution d'une transaction, la vue change, cette transaction doit être réexécutée avec la nouvelle vue, afin de satisfaire la règle d'unicité de vue (une transaction doit être exécutée dans une seule et même vue). Il faut alors former la nouvelle vue puis mettre à niveau les copies de cette vue. Le protocole de **formation d'une nouvelle vue** comprend deux phases. La première phase est une phase d'invitation à participer à la nouvelle vue. Seuls les processeurs acceptant l'invitation acquittent. La deuxième phase fournit la composition de la nouvelle vue. Les copies de la nouvelle vue sont ensuite **mises à niveau** à l'aide d'une transaction spécifique : lecture de *QL* copies et écriture de la version la plus à jour sur toutes les copies de la vue.

Les algorithmes dits **à consensus unanime** appartiennent à cette sous-classe. Cette approche a inspiré les algorithmes dits à synchronie virtuelle [Birman, Joseph 1987], lesquels ont néanmoins été présentés pour un modèle DNB, ce qui pose le problème de leur terminaison certaine (voir [Fischer, Lynch, Paterson 1985]).

c) Algorithmes à quorum dynamique

L'algorithme **à quorum dynamique** proposé dans [Bhargava, Browne 1990] constitue une généralisation des deux sous-classes précédentes : il utilise les concepts de vue et de quorum de copies. La vue est alors définie comme l'ensemble des copies utilisant une même assignation de quorums. La règle d'unicité de vue doit être satisfaite pour chaque transaction. Cet algorithme permet une modification dynamique des assignations de quorums (avec ou sans formation d'une nouvelle vue selon la disponibilité des quorums) et la réutilisation après recouvrement des quorums non modifiés lors de la défaillance. Le recouvrement dépend ainsi du profil des transactions exécutées durant la défaillance.

L'algorithme de **quorum en arbre** décrit dans [Agrawal, El Abbadi 1990] est très flexible et présente l'avantage de ne requérir aucune coordination entre les copies lors d'un changement d'accessibilité de l'une d'entre elles. Il est particulièrement adapté à la gestion d'un grand nombre de copies. Il fait l'hypothèse d'une organisation des copies selon un arbre n-aire équilibré. Cet arbre, connu de chaque copie, est utilisé pour déterminer les quorums. Un quorum de lecture est formé par la copie racine et si la racine est inaccessible, elle est remplacée par une majorité de ses enfants et ainsi de suite. Un quorum d'écriture est formé par la copie racine, une majorité de ses enfants et pour chaque enfant sélectionné : une majorité de ses enfants et ainsi de suite. En fonctionnement normal, une lecture se traduit par la lecture de la copie racine, une écriture se traduit par l'écriture d'une majorité de copies à chaque niveau de l'arbre. La structuration des copies en arbre a cependant deux inconvénients : la racine de l'arbre

constitue un goulot d'étranglement et un point dur vis-à-vis de la disponibilité en écriture.

Un autre algorithme bien adapté à la gestion d'un grand nombre de copies est l'algorithme en **grille** dans [Cheung, Ammar, Ahamad 1992]. Cet algorithme est basé sur une organisation des copies en une grille logique de *l* lignes et *c* colonnes. Un quorum de lecture consiste en une seule copie de chaque colonne, tandis qu'un quorum d'écriture consiste en toutes les copies d'une colonne plus une copie de chacune des autres colonnes.

5. Diffusion

Comme nous venons de le voir, les algorithmes de gestion des copies multiples ont besoin de diffuser les mises-à-jour sur les différentes copies. Le protocole de validation en deux phases utilise lui aussi la diffusion. La diffusion est également utilisée dans la réplication de processus :

- en réplication active : diffusion des entrées aux répliques,
- en réplication passive : diffusion des points de reprise aux secondaires,
- en réplication semi-active : diffusion des entrées / notifications aux répliques suiveuses.

Plus généralement une diffusion est utilisée lorsqu'un processeur veut faire connaître à plusieurs processeurs une information qu'il détient localement. Nous examinons dans ce paragraphe différents types de diffusion et concluons par les relations entre diffusion atomique, appartenance à un groupe et consensus.

5.1. Principe

La source de la diffusion diffuse un message (son avis) à tous les processeurs appartenant au groupe de diffusion. Selon l'issue de la diffusion (décision imposée par la source), les processeurs du groupe remettent ou rejettent le message reçu. Le problème de la diffusion n'est présenté que dans le cadre du modèle DBC. Ce problème se définit par :

- la nature de l'accord obtenu entre tous les membres du groupe,
- l'ordre de remise des messages par les différents membres du groupe.

Relativement au critère d'**ordre**, il existe au moins trois possibilités :

- **aucune garantie** : les membres corrects du groupe de diffusion remettent les messages dans n'importe quel ordre ;
- **garantie de l'ordre causal** : l'ordre de remise des messages par les membres corrects reflète l'ordre causal de leur émission ;
- **garantie de l'ordre total** : l'ordre de remise des messages est le même pour tous les membres corrects.

Ces définitions font apparaitre la notion de membre correct, qui fait référence aux hypothèses de fautes tolérées par le protocole de diffusion. Remarquons qu'elles ne spécifient en rien le comportement des membres incorrects.

Relativement au critère **nature de l'accord obtenu**, il existe cinq possibilités :

- **aucune garantie** : un membre correct du groupe de diffusion remet le message diffusé tandis qu'un autre membre correct peut rejeter ce même message ;
- **garantie d'un accord partiel** : si un membre correct remet un message, alors au moins *k* membres du groupe ont été d'accord pour remettre ce message ;
- **garantie d'unanimité des corrects** : si un membre correct remet un message, alors tous les membres corrects du groupe remettent le message ;
- **garantie d'uniformité** : si un membre correct ou incorrect remet un message, alors tous les membres corrects remettent ce message ;
- **garantie d'unanimité globale** : si un membre correct ou incorrect remet un message, alors tous les membres (tous et non pas seulement les membres corrects) remettent ce message. Si par contre un membre est dans l'impossibilité de remettre un message (ex. : membre isolé ou membre défaillant), alors aucun membre ne remet ce message.

De même remarquons que les trois premières définitions ne prennent pas en compte le comportement des membres incorrects; elles ne sont pertinentes que lorsque les processeurs sont amnésiques lors d'un recouvrement. L'unanimité globale exige que les incorrects remettent (lors de leur recouvrement, s'ils n'ont pu le faire avant) les seuls messages remis par les corrects et rejettent les seuls messages rejetés par les corrects.

De plus on peut définir la propriété de préfixe. La **propriété de préfixe** [Anceaume , Minet 1993] est vraie si la séquence des messages remis par un membre incorrect est un préfixe de la séquence des messages remis par les membres corrects.

5.2. Types de diffusion

On connaît les types de diffusion suivants :

- **diffusion au mieux (best effort)** : aucun compte-rendu n'est fourni, le fournisseur de service fait de son mieux mais n'apporte aucune garantie ;
- **diffusion à unanimité globale** : le message n'est remis que dans la mesure où tous sont d'accord, il est annulé sinon. Les incorrects doivent remettre les seuls messages remis par les corrects et rejeter les seuls messages rejetés par les corrects. Certaines situations pouvant conduire à une indécidabilité temporaire (cf protocole de validation en deux phases) ;
- **diffusion fiable** : cette diffusion garantit l'unanimité des corrects mais n'offre aucune garantie quant à l'ordre. Une diffusion fiable est utilisée lorsqu'il n'existe qu'une seule source de diffusion possible à travers le groupe de diffusion [Powell 1991], ou encore lorsque l'ordre de remise des messages n'a aucune importance ;
- **diffusion causale** : cette diffusion garantit l'unanimité des corrects et l'ordre causal [Birman, Schiper, Stephenson 1990]. Les solutions réalisées favorisent les performances et peuvent conduire à des scénarios (défaillance successive de deux processeurs) où un message est remis par les corrects alors que le message dont il dépend causalement a été perdu.
- **diffusion atomique** : cette diffusion garantit l'unanimité des corrects et l'ordre total. Elle est utilisée pour diffuser les mises-à-jour d'une variable globale, dont

chaque processeur maintient une copie ([Anceaume 1993], [Chang, Maxemchuk 1984], [Cristian 1990], [Cristian, Aghali, Strong, Dolev 1985], [Gligor, Luan 1990], [Gopal, Toueg 1989], [Kaashoek, Tanenbaum, Hummel, Bal 1989], [Veríssimo, Rodrigues, Baptista 1989]).

• **diffusion atomique et causale** : cette diffusion est atomique et l'ordre total de remise des messages reflète l'ordre causal de leur émission (ex: xAMp de Delta4 [Powell 1991], ABCAST de ISIS [Birman, Schiper, Stephenson 1990]).

Pour un type de diffusion donné, les différents protocoles se distinguent par les hypothèses adoptées relativement :

• **aux membres du groupe** : mode de défaillance supposé, nombre maximum de défaillances tolérées ...

• **au système de communication sous-jacent** : mode de défaillance supposé, qualité du service de communication (ex1 : tout message émis par un membre correct est reçu par au moins p membres corrects, ex2 : tout message émis par un membre correct au temps T est reçu par tout membre correct au plus tard au temps $T+ dmax$ [Cristian 1990]) ...

• **aux horloges utilisées** : une horloge globale [Gopal, Toueg 1989], des horloges locales [Anceaume 1993], [Cristian 1990].

5.3. Appartenance au groupe de diffusion

Ces types de diffusion font entrevoir trois possibilités vis-à-vis des membres incorrects :

• **une première possibilité consiste à les ignorer totalement dans l'obtention des accords**. Dans les protocoles de diffusion de cette catégorie [Cristian 1990], un membre correct ne connaît pas les autres membres corrects du groupe. La connaissance des membres corrects du groupe est acquise à un niveau supérieur par un protocole d'appartenance au groupe [Cristian 1988] ;

• **une deuxième possibilité consiste à exclure du groupe de diffusion les membres incorrects et donc à les ignorer totalement dans l'obtention des accords ultérieurs** (ceci jusqu'à leur retour à l'état opérationnel et leur réinsertion dans le système). Les protocoles de cette famille intègrent dans le protocole de diffusion un protocole d'appartenance au groupe ([Anceaume 1993], [Chang, Maxemchuk 1984], [Kaashoek, Tanenbaum, Hummel, Bal 1989], [Veríssimo, Rodrigues, Baptista 1989]). Ainsi au niveau du protocole de diffusion, tout membre correct connaît les autres membres corrects du groupe. Généralement c'est la source de diffusion qui déclenche l'exclusion d'un membre sur non-réception d'un acquittement de ce dernier. Signalons le risque d'exclusion abusive : c'est-à-dire l'exclusion d'un membre correct par un membre incorrect [Anceaume 1993].

• **une troisième possibilité consiste à annuler tous les messages qui ne parviennent pas à réunir l'accord de tous les membres**. C'est l'approche retenue dans le protocole de validation en deux phases en l'absence de copies multiples. Les transactions ne faisant pas intervenir un membre défaillant peuvent bien évidemment s'exécuter.

6. Consensus distribué tolérant aux fautes

Le problème du consensus est très lié au problème de la diffusion. En diffusion, un processeur communique une connaissance à d'autres processeurs. En consensus, chaque processeur a une connaissance locale ou un avis initial. Le but est d'obtenir un avis unique (algorithme de décision) qui représente l'avis du groupe. Le consensus utilise la diffusion, et souvent les problèmes de diffusion sont équivalents à des problèmes de consensus.

6.1. Le problème du consensus unanime

Soit n processeurs. Soit $v(i)$ l'avis initial du processeur i. Soit $W(i)$ le vecteur reflétant la connaissance qu'a le processeur i des avis des autres (un protocole de diffusion est souvent utilisé pour faire connaître les avis des processeurs). Certains processeurs et le système de communication peuvent être fautifs. On dit qu'il y a **consensus** si et seulement si les propriétés suivantes sont obtenues :

(1) pour 2 processeurs i et j corrects, on a : $W(i) = W(j) = [w1, w2, ..., wn]$

(2) si k est un processeur correct, alors on a : $wk = v(k)$

NB: si k est fautif, wk n'est pas spécifié mais doit être unique pour tous les processeurs corrects.

Exemples de consensus réparti : reconfiguration dynamique, appartenance de groupe (membership), temps global.

Les algorithmes de décision utilisés dépendent de la nature du consensus à obtenir. S'il s'agit de booléens par exemple, on ne peut appliquer des décisions basées sur des calculs de moyennne, comme c'est le cas par exemple avec des valeurs de capteurs ou d'horloges.

6.2. Solutions

Les solutions au problème de consensus dépendent du modèle conceptuel temporel retenu (cf §2.1.).

a) Modèle DBC

On trouve sous ce modèle les solutions les plus connues. Ces solutions exploitent la connaissance des valeurs des bornes sur les délais et du temps global (cf paragraphe suivant), à la manière de l'approche "state-machine" [Schneider 1990], [Lamport 1984] qui consiste à attendre au moins l'heure $t + max$ avant de traiter un événement apparu à l'heure t. Il ne peut ainsi y avoir d'inversion d'ordre entre événements.

Un exemple célèbre de consensus réparti en modèle DBC est l'algorithme dit des "messages oraux" de résolution du problème des Généraux Byzantins [Lamport, Shostak, Pease 1982].

Cependant, ces solutions sont vulnérables aux défaillances temporelles de la communication, car le fait que la précision du temps global ne peut être nulle, interdit de supposer que les tests de ponctualité seront identiques chez tous les processeurs.

[Chandy, Lamport 1985] K. M. Chandy and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems", *ACM Trans. Computer Systems*, 3 (1), pp.63-75, February 1985.

[Chandra, Toueg 1991] T. Chandra, S. Toueg, "Unreliable failure detectors for asynchronous systems", 10th ACM Symposium on Principles of Distributed Computing,ACM Press, pp 325-340, August 1991.

[Chang, Maxemchuk 1984] J. Chang and N. Maxemchuk, "Reliable Broadcast Protocols", *ACM Trans. Computer Systems*, 2 (3), pp.251-273, August 1984.

[Chérèque et al 1992] M. Chérèque, D. Powell, P. Reynier, J.-L. Richier and J. Voiron, "Active Replication in Delta-4", in *Proc. 22nd Int. Conf. on Fault-Tolerant Computing Systems (FTCS-22)*, (Boston, MA, USA), pp.28-37, IEEE Computer Society Press, July 1992.

[Cheung, Ammar, Ahamad 1992] S. Y. Cheung, M. H. Ammar and M. Ahamad, "The Grid Protocol: a High Performance Scheme for Maintaining Replicated Data", *IEEE Transactions on Knowledge and Data Engineering*, 4 (6), pp.582-592, December 1992.

[Cristian 1988] F. Cristian, "Agreeing on Who is Present and Who is Absent in a Synchronous Distributed System", in *Proc. 18th Int. Symp. on Fault-Tolerant Computing (FTCS-18)*, (Tokyo, Japan), pp.206-211, IEEE Computer Society Press, June 1988.

[Cristian 1989] F. Cristian, "Probabilistic Clock Synchronization", *Distributed Computing*, 3 (3), pp.146-158, 1989.

[Cristian 1990] F. Cristian, "Synchronous Atomic Broadcast for Redundant Broadcast Channels", *J. Real-Time Systems*, 2, pp.195-212, 1990.

[Cristian 1991] F. Cristian, "Understanding Fault-Tolerant Distributed Systems", *Comm. ACM*, 34 (2), pp.56-78, February 1991.

[Cristian, Aghali, Strong, Dolev 1985] F. Cristian, H. Aghali, R. Strong and D. Dolev, "Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement", in *Proc. 15th Int. Symp. on Fault-Tolerant Computing (FTCS-15)*, (Ann Arbor, MI, USA), pp.200-206, IEEE Computer Society Press, June 1985.

[Dolev, Dwork, Stockmeyer 1987] D. Dolev, C. Dwork and L. Stockmeyer, "On the Minimal Synchronism needed for Distributed Consensus", *Journ. of the ACM*, 34 (1), pp.77-97, January 1987.

[Dwork, Lynch, Stockmeyer 1988] C. Dwork, N. Lynch and L. Stockmeyer, "Consensus in the Presence of Partial Synchrony", *Journ. of the ACM*, 35 (2), pp.288-323, April 1988.

[Eager, Sevcik1983] D. L. Eager and K. C. Sevcik, "Achieving Robustness in Distributed Database Systems", *ACM Trans. on Database Sys.*, 8 (3), pp.354-381, September 1983.

[El Abbadi, Skeen, Cristian 1985] A. El Abbadi, D. Skeen and F. Cristian, "An Efficient Fault-Tolerant Protocol for Replicated Data Management", in *Proc. 4th ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, (Portland, OR, USA), pp.215-228, ACM, March 1985.

[Fischer 1990] M. Fischer, "A Theoretician's View of Fault Tolerant Distributed Computing", in *Fault-Tolerant Distributed Computing*, (B. Simons and A. Spector, Ed.), Lecture Notes on Computer Science, 448, pp.1-9, Springer-Verlag, Berlin, Germany, 1990.

[Fischer, Lynch, Paterson 1985] M. Fischer, N. Lynch, M. Paterson, "Impossibility of Distributed Consensus with One Faulty Processor", jal of ACM, 32(2), 1985.

[Gifford 1979] D. K. Gifford, "Weighted Voting for Replicated Data", in *Proc. 7th Symp. on Operating System Principles*, (Asilomar, CA, USA), pp.150-162, December 1979.

[Gligor, Luan 1990] V. Gligor and W. Luan, "A Fault-Tolerant Protocol for Atomic Broadcast", *IEEE Trans. Parallel and Distributed Systems*, 1 (3), pp.271-285, July 1990.

[Gopal, Strong, Toueg, Cristian 1990] A. Gopal, R. Strong, S. Toueg and F. Cristian, "Early Delivery Atomic Broadcast", in *Proc. 9th Symp. on Principles of Distributed Computing*, pp.297-310, ACM, August 1990.

[Gopal, Toueg 1989] A. Gopal and S. Toueg, "Reliable Broadcast in Synchronous and Asynchronous Environments", in *Proc. 3rd Int. Workshop on Distributed Algorithms*, (Nice, France), September 1989.

[Herlihy 1986] M. Herlihy, "A Quorum Consensus Replication Method for Abstract Data Types", *ACM Trans. Computer Systems*, 4 (1), pp.32-53, February 1986.

[ISO/IEC 1992] *Transactional Processing*, ISO/IEC N IS10026 (April 1992).

[Johnson, Zwaenepoel 1990] D. B. Johnson and W. Zwaenepoel, "Recovery in Distributed Systems Using Optimistic Message Logging and Checkpointing", *Journ. of Algorithms*, 11, pp.462-491, 1990.

[Kaashoek, Tanenbaum, Hummel, Bal 1989] F. Kaashoek, S. Tanenbaum, F. Hummel and E. Bal, "An Efficient Reliable Broadcast Protocol", *ACM Op. Sys. Review*, 23 (4), pp.5-19, October 1989.

[Koo, Toueg 1987] R. Koo and S. Toueg, "Checkpointing and Rollback Recovery for Distributed Systems", *IEEE Trans. Software Engineering*, SE-13 (1), pp.23-31, January 1987.

[Kopetz, Ochsenreiter 1987] H. Kopetz and W. Ochsenreiter, "Clock Synchronization in Distributed Real-Time Systems", *IEEE Trans. Computers*, C-36 (8), pp.933-940, August 1987.

[Lamport 1978] L. Lamport, "Time, Clocks and the Ordering of Events in a Distributed System", Communication of the ACM, 21 (7), pp.558-565, July 1978.

Lamport 1984] L. Lamport, "Using Time instead of Timeout for Fault-Tolerant Distributed Systems", ACM Transactions on Programming Languages and Systems, 6 (2), pp.254-280, April 1984.

[Lamport, Melliar-Smith 1984] L. Lamport and P. Melliar-Smith, "Byzantine Clock Synchronization", in *Proc. 3rd Symp. on Principles of Distributed Computing*, (Vancouver, Canada), pp.68-74, ACM, August 1984.

[Lamport, Shostak, Pease 1982] L. Lamport, R. Shostak and M. Pease, "The Byzantine Generals Problem", *ACM Trans. Programming Languages and Systems*, 4 (3), pp.382-401, July 1982.

[Lampson 1981] B. W. Lampson, "Atomic Transactions", in *Distributed Systems — Architecture and Implementation,* (B. W. Lampson, Ed.), Lecture Notes in Computer Science, 105, Springer-Verlag, Berlin, Germany, 1981.

[Laprie 1992] J.-C. Laprie (Ed.), *Dependability: Basic Concepts and Terminology,* Dependable Computing and Fault-Tolerance, 5, 265p., Springer-Verlag, Vienna, Austria, 1992.

[Le Lann 1977] G. Le Lann, "Distributed Systems :Towards a Formal Approach", IFIP Congress, Toronto Canada, North Holland, pp. 155-160, June 1977.

[Le Lann 1981] G. Le Lann, "A Distributed System for Real-Time Transaction Processing", IEEE Computer, 14 (2), pp.43-48, February 1981.

[Le Lann 1992] G. Le Lann, "Designing Real-Time Dependable Distributed Systems", *Jal of Computer Communications*, Butterworth-Heinemann pub., 15 (4), pp. 225-234, , May 1992

[Leu, Bhargava 1989] P. Leu and B. Bhargava, "A Model for Concurrent Checkpointing and Recovery Using Transactions", in *Proc. 9th Int. Conf. on Distributed Computing Systems (ICDCS-9),* (Newport Beach, CA, USA), pp.423-430, IEEE Computer Society Press, June 1989.

[Lindsay et al 1979] B. Lindsay, P. Sellinger, C. Galtieri, J. Gray, R. Lorie, T. Price, G. Putzolu, L.Traiger, B. Wade, "Notes on Distributed Databases", IBM San Jose Research Lab., RJ 2571 (33471), 1979.

[Lundelius-Welch, Lynch 1988] J. Lundelius-Welch and N. Lynch, "A New Fault-Tolerant Algorithm for Clock Synchronization", *Information & Computation*, 77 (1), pp.1-16, 1988.

[Mishra, Schlichting 1992] S. Mishra and R. Schlichting, *Abstractions for Constructing Dependable Distributed Systems*, Dept. of Computer Science, University of Arizona, AZ, USA, Report, N TR 92 -12, August 1992.

[Mullender 1989] S. Mullender (Ed.), *Distributed Systems,* ACM Press, Addison-Wesley, New York, 1989.

[Powell 1991] D. Powell (Ed.), *Delta-4: a Generic Architecture for Dependable Distributed Computing,* Research Reports ESPRIT, 484p., Springer-Verlag, Berlin, Germany, 1991.

[Powell 1992] D. Powell, "Failure Mode Assumptions and Assumption Coverage", in *Proc. 22nd Int. Symp. on Fault-Tolerant Computing (FTCS-22),* (Boston, MA, USA), pp.386-395, IEEE Computer Society Press, July 1992.

[Powell et al 1988] D. Powell, G. Bonn, D. Seaton, P. Veríssimo and F. Waeselynck, "The Delta-4 Approach to Dependability in Open Distributed Computing Systems", in *Proc. 18th Int. Symp. on Fault-Tolerant Computing Systems (FTCS-18),* (Tokyo, Japan), pp.246-251, IEEE Computer Society Press, June 1988.

[Randell 1975] B. Randell, "System Structure for Software Fault Tolerance", *IEEE Trans. Software Engineering*, SE-1 (2), pp.220-232, 1975.

[Raynal 1991] M. Raynal, *La communication et le temps dans les réseaux et les systèmes répartis,* Introduction aux principes algorithmiques des systèmes répartis, 1, 220p., Eyrolles, Paris, 1991.

[Schlichting, Schneider 1983] R. D. Schlichting and F. B. Schneider, "Fail-Stop

Processors: An Approach to Designing Fault-Tolerant Computing Systems", *ACM Trans. Computer Systems*, 1 (3), pp.222-238, August 1983.

[Schneider 1984] F. B. Schneider, "Byzantine Generals in Action: Implementing Fail-Stop Processors", *ACM Trans. Computer Systems*, 2 (2), pp.145-154, May 1984.

[Schneider 1990] F. B. Schneider, "Implementing Fault Tolerant Services using the State Machine Approach: a Tutorial", *ACM Comp. Surveys*, 22 (4), pp.229-319, December 1990.

[Shrivastava, Mancini, Randell 1987] S. K. Shrivastava, L. V. Mancini and B. Randell, "On the Duality of Fault-Tolerant System Structures", in *Experiences with Distributed Systems*, (J. Nehmer, Ed.), Lecture Notes in Computer Science, 309, pp.19-37, Springer-Verlag, 1987.

[Spector 1989] A. Z. Spector, "Distributed Transaction Processing Facilities", in *Distributed Systems*, (S. Mullender, Ed.), pp.191-214, ACM Press, Addison-Wesley, New York, 1989.

[Srikanth, Toueg 1987] T. K. Srikanth and S. Toueg, "Optimal Clock Synchronization", *Journal of the Association for Computing Machinery*, 34 (3), pp.626-645, July 1987.

[Stonebraker 1979] M. Stonebraker, "Concurrency Control in Distributed Ingres", *IEEE Trans. Software Engineering*, 5 (3), pp.188-194, May 1979.

[Strom, Yemini 1985] R. E. Strom and S. Yemini, "Optimistic Recovery in Distributed Systems", *ACM Trans. Computer Systems*, 3 (3), pp.204-226, August 1985.

[Thomas 1979] R. H. Thomas, "A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases", *ACM Trans. on Database Sys.*, 4 (2), pp.180-209, June 1979.

[Veríssimo, Rodrigues, Baptista 1989] P. Veríssimo, L. Rodrigues and M. Baptista, "AMp: A Highly Parallel Atomic Multicast Protocol", *ACM Comp. Comm. Review*, 19 (4), pp.83-93, September 1989.

Unité de Recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)

Unité de Recherche INRIA Lorraine Technopôle de Nancy-Brabois - Campus Scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 VILLERS LES NANCY Cedex (France)
Unité de Recherche INRIA Rennes IRISA, Campus Universitaire de Beaulieu 35042 RENNES Cedex (France)
Unité de Recherche INRIA Rhône-Alpes 46, avenue Félix Viallet - 38031 GRENOBLE Cedex (France)
Unité de Recherche INRIA Sophia Antipolis 2004, route des Lucioles - B.P. 93 - 06902 SOPHIA ANTIPOLIS Cedex (France)

EDITEUR
INRIA - Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)

ISSN 0249 - 6399