Problèmes de communication et de coordination dans les systèmes spatiaux

Gerard Le Lann

Problèmes de communication et de coordination dans les systèmes spatiaux

Gérard Le Lann

INRIA Rocquencourt
BP 105
F-78153 Le Chesnay cedex, France
Gerard.Le_Lann@inria.fr

RÉSUMÉ. *Les problèmes de communication et ceux de coordination peuvent être intimement liés dans le cas des systèmes spatiaux, à cause notamment de l'existence de défaillances arbitraires et des exigences strictes de haute sûreté de fonctionnement. On donne une caractérisation du domaine spatial, tant du point de vue « communications » que du point de vue « traitements ». Des problèmes génériques majeurs combinant le distribué, le temps réel et la tolérance aux fautes sont examinés et illustrés par des analyses de problèmes vécus lors de missions spatiales.*

ABSTRACT. *Communication problems and coordination problems may turn out to be intimately related in space systems, notably due to the need to cope with arbitrary failures, as well as strict dependability requirements. A characterization of the space domain is given, from the "communication" angle, as well as from the "computing" angle. Major generic problems combining distribution, real-time, and fault-tolerance issues are examined and illustrated with analyses of problems experienced in the course of space missions.*

MOTS-CLÉS : *système spatial, système distribué, réseau de mobiles, réseau interplanétaire, TCP, sûreté de fonctionnement, temps réel, terminaison atomique non bloquante, accord approché, consensus.*

KEYWORDS: *space system, distributed system, network of mobiles, interplanetary network, TCP, dependability, real-time, non blocking atomic commit, approximate agreement, consensus.*

Nom de la revue. Volume X – n° X/2001, pages 1 à X

1. Introduction

Dans le domaine spatial, tout système comprend :

- un segment sol, situé sur la planète Terre, composé de sites statiques (dans des bâtiments) ou mobiles (navires par exemple), intégrant des opérateurs humains (contrôleurs de missions),

- un segment espace, composé d'entités mobiles (satellites, sondes, véhicules spatiaux, stations orbitales) situées à des distances (constantes ou variables) connues de la Terre.

Continûment ou bien pour certaines phases de missions, les entités mobiles fonctionnent soit avec des opérateurs humains (équipage de l'ISS—station spatiale internationale), soit de façon quasi-autonome (rovers sous le contrôle d'un orbiteur ou de la Terre), soit de façon totalement autonome. L'autonomie totale (permanente ou temporaire) est requise notamment pour les sondes d'exploration lointaine (« deep space »), la rentrée en atmosphère terrestre des véhicules habités (« Space Shuttle » (USA), « Autonomous Transfer Vehicle » (EU)), la mise en orbite autour d'une planète, le largage et le déploiement de rovers sur le sol d'une planète, ou, dans un avenir proche, les manœuvres de rendez-vous et d'arrimage, les constellations de satellites, les flottilles de sondes spatiales.

Pour les besoins de cet article, un système spatial, noté Σ, peut être vu comme étant structuré en trois sous-systèmes[1] :

- les équipements informatiques (calculateurs & périphériques du segment sol, processeurs et mémoires de stockage embarqués espace, leurs logiciels applicatifs et système), sous-système noté S,

- les équipements de communication (réseaux essentiellement câblés pour le segment sol, constitués de liaisons radio, infra-rouge, optique pour le segment espace, leurs protocoles), sous-système noté R,

- les équipements de contrôle-suivi spécifiques aux missions spatiales (caméras, télescopes, capteurs et actionneurs divers, gyromètres, antennes, etc., leurs logiciels), sous-système noté C.

Cet article traite uniquement des problèmes de niveau *système*, ceux posés par l'obligation de prédire et garantir que les comportements *globaux* de Σ, ou de sous-ensembles de Σ, sont satisfaisants. Les problèmes de niveau *local* à chacun des équipements ne sont pas abordés. Ainsi, la gestion des communications internes à un satellite n'entre pas dans le champ de cet article.

Dans le cas des applications reposant sur des systèmes non mobiles et/ou des réseaux câblés, il est établi (cf. état de l'art académique et industriel) qu'il est

[1] Par manque de place, on ignore dans cet article le sous-système composé d'opérateurs humains (qui pose également des problèmes de niveau *système*).

possible et recommandé de séparer les problèmes relevant de S, ou bien de R, ou bien de C. Depuis une décennie environ, l'on sait que cette décomposition stricte n'est plus valide pour les systèmes reposant sur des réseaux non filaires et/ou mobiles, le cas pour les systèmes spatiaux. En conséquence, l'article est structuré en problèmes/solutions génériques et études de cas, plutôt qu'en une décomposition « réseau » (communications) et « traitement » (exécutions de programmes).

Avec les systèmes spatiaux, l'on est confronté à l'obligation de satisfaire plusieurs exigences contradictoires, au rang desquelles l'on trouve, en particulier, une exigence de très haute sûreté de fonctionnement—notée SdF, malgré des conditions opérationnelles particulièrement « agressives ». La SdF est requise pour de multiples raisons, telles la sûreté (préserver les vies des astronautes) ou l'efficacité. Perdre une mission d'exploration sur Mars coûte le prix des équipements perdus (de l'ordre de 100 à 300 M€) plus le prix de la mission elle-même (préparation, non récupération des informations espérées, etc.).

Comme c'est souvent le cas avec les applications innovantes, le domaine spatial met en évidence des ruptures de connaissances. C'est en particulier le cas pour la SdF, tant dans le milieu de la recherche que dans celui de l'industrie. En simplifiant, on constate que les modèles de fautes et de défaillances (causées par les fautes) auxquels l'on est confronté dans le spatial sont :

- connus et traités par la communauté SdF « traditionnelle » (« stuck-at-one, stuck-at-zero », systèmes de type monoprocesseur, etc.), qui dispose de solutions *locales* à un équipement,

- ignorés par la communauté « méthodes formelles » (langages synchrones, « model checking », etc.), car non exprimables dans les formalismes connus qui se prêtent aux vérifications mécanisées[2],

- ignorés par une grande fraction de la communauté « systèmes distribués » qui, jusqu'à récemment, a traité exclusivement les défaillances *permanentes*, lesquelles ne correspondent pas à la catégorie dominante des défaillances propres au spatial, à savoir les défaillances transitoires et/ou intermittentes ; de plus, jusqu'à récemment, les travaux de cette communauté ont porté exclusivement sur les systèmes statiques/câblés. En conséquence, les solutions *globales* (système) générées par cette communauté sont trop souvent inapplicables (telles quelles ou « adaptées ») aux systèmes spatiaux.

Les problèmes posés par la conception et la validation des *systèmes* (systèmes spatiaux ou autres systèmes) sont différents des problèmes posés par la conception et la validation des *logiciels*, y compris ceux qui servent à implanter des spécifications d'algorithmes « système » (Le Lann, 1998). De même, les problèmes posés par la conception et la validation d'un plan de barrage sont différents de ceux posés par la conception et la validation des blocs de béton (qui servent à implanter un plan de

[2] Pour une mise en perspective de ces méthodes, voir (Bowen *et al*., 2006).

barrage). Une fausse « vérité » entretenue depuis fort longtemps est que « seul le logiciel compte dans les systèmes informatiques ». Il suffit de remarquer que le logiciel est une technologie d'implantation, au même titre que l'électronique ou l'optique. Mais avant d'implanter, il faut bien évidemment savoir *quoi* implanter. Il est donc nécessaire d'établir au préalable une spécification prouvée correcte (en regard d'une spécification du problème applicatif considéré) du système qui doit être implanté. Il est évident que les connaissances requises pour construire et valider une spécification de $\Sigma = S \cup R \cup C$ ne relèvent pas de l'ingénierie logicielle, mais de l'ingénierie système.

La plupart des problèmes de SdF propres au spatial sont dans l'intersection des trois domaines « systèmes distribués », « tolérance aux fautes », « temps réel », les solutions devant être accompagnées de leurs preuves de propriétés. Etant donné les sémantiques de modélisation correcte des systèmes spatiaux (par exemple, modèles de fautes et défaillances allant jusqu'au modèle byzantin, tous les modèles de système/calcul à l'exclusion, presque toujours, du modèle synchrone pur[3], modèles de processus et ressources partagées à états modifiables rémanents, concomitance et conflits d'accès, propriétés exigées telles la minimisation des temps de réponse en présence de surcharges ou la diffusion atomique), il est évident que seules les preuves au sens classique des Mathématiques peuvent être envisagées, leur « formalisation » (mécanisation) ne pouvant survenir que dans un avenir mal déterminé (Hoare, 2003). Pour les sous-systèmes R (la communauté « réseaux et protocoles »), cette évidence est entérinée depuis fort longtemps[4]. Pour les sous-systèmes composites $S \cup C$, c'est une communauté restreinte qui génère actuellement des solutions pour ces problèmes[5].

Le double but de cet article est d'illustrer ces observations et de caractériser certains des problèmes majeurs. Un troisième but, indirect celui-là, est de susciter des vocations (de recherche pure et de R&D) ayant pour finalité de résoudre des problèmes ouverts. Le spatial partage de nombreux problèmes avec la plupart des domaines applicatifs où l'autonomie est combinée à une exigence de très haute sûreté de fonctionnement. Les sujets abordés dans cet article sont donc porteurs d'avenir.

On donne tout d'abord une caractérisation du domaine spatial (§2). Ensuite, des problèmes majeurs combinant le « distribué », le « temps réel » et la « tolérance aux fautes » sont examinés et illustrés par des études de cas (§3). A chaque fois, on analyse ce qui relève des communications (« networking ») et ce qui relève des traitements (« computing »).

[3] Les raisons en sont données plus loin (§2).
[4] Voir par exemple les publications de la revue ACM/IEEE Networking.
[5] En simplifiant, il s'agit de la communauté ayant la double appartenance ACM PODC (Principles of Distributed Computing) et IEEE RTSS (Real-Time Systems Symposium).

2. Caractéristiques propres au domaine spatial

On distingue les modèles/hypothèses (« l'axiomatique » environnementale) des propriétés exigées (les services et QoS devant être démontrés).

2.1. *Conditions environnementales*

2.1.1. *Communications et réseaux (sous-systèmes R)*

Un réseau sol/espace ou espace-espace, tel l'Internet Interplanétaire—IPN Internet (Cerf *et al.*, 2001), est de type hybride (filaire, non filaire). Les distances élevées entraînent de grands délais de propagation. De plus, pour cause de mobilité des véhicules spatiaux … et des planètes, ces délais sont très variables. Ainsi, les délais d'aller-retour Terre-Mars varient entre 8,5 et 40 minutes, et entre 593,3 et 1044,4 minutes pour les allers-retours Terre-Jupiter, selon les positions respectives des planètes.

Les sources d'énergie étant en général limitées dans le segment espace, tandis qu'elles ne le sont pas dans le segment sol, les vitesses nominales de transmission sont très variées, et peuvent être faibles dans le segment espace. En outre, les capacités des liaisons espace-sol et sol-espace sont fondamentalement dissymétriques. La mobilité, la directivité des antennes, le milieu physique (cf. plus loin) entraînent des taux élevés d'erreurs de transmission. Par exemple, le taux d'erreur bit de l'ossature (« backbone ») de l'Internet Interplanétaire est de l'ordre de 10^{-1} (Akyildiz *et al.*, 2004).

Enfin, un réseau sol/espace ou espace-espace est sujet à déconnexions et reconnexions programmées (mouvements connus des planètes, des satellites terrestres, etc.) ou bien intempestives (ruptures accidentelles, intermittentes, de liaisons). Les partitionnements de réseaux, notamment ceux qui isolent le segment espace du segment sol, sont un sujet de préoccupation majeure.

Sauf cas particuliers (certains réseaux « très » locaux), les réseaux connus sont du type « store-and-forward ». Les réseaux filaires terrestres classiques (Internet, etc.) sont de type s&F : les nœuds font du « store » bref et du « forward » rapide, car le taux de disponibilité des routes (à capacités élevées) est très élevé. A l'inverse, les réseaux sol/espace sont de type S&f : les nœuds sont amenés à faire du « store » prolongé et du « forward » lent, car le taux de disponibilité des routes (à capacités limitées) peut être faible.

On voit donc que ces réseaux présentent de nombreuses similitudes avec les réseaux de type ad hoc sans ossature (mobiles—cf. MANET, statiques—cf. SANET), avec ossature—cf. « meshed ad hoc networks », ainsi qu'avec les réseaux de type DTN (« delay tolerant networks ») et les réseaux de capteurs (« sensor networks »).

Les protocoles qui régissent les comportements des sous-systèmes R et les algorithmes système (non applicatifs) qui régissent les comportements des sous-

systèmes S ∪ C ont de nombreux attributs communs. Le travail de conception, de preuve et d'évaluation des performances d'un algorithme de routage et celui relatif à un algorithme de consensus distribué sont quasiment identiques, même si les approches relèvent tantôt du « stochastique », tantôt du « déterministe ». Une différence majeure existe cependant, d'ordre normatif : la plupart des logiciels/matériels qui implantent des protocoles destinés aux sous-systèmes R sont soumis à normalisation, ce qui n'est pas le cas pour les logiciels/matériels qui implantent des algorithmes applicatifs ou système destinés aux sous-systèmes S ∪ C. Par contre, certains de ces derniers sont soumis à certification (cf. plus loin).

2.1.2. *Traitements (sous-systèmes S ∪ C)*

Dans un véhicule spatial, il existe des logiciels applicatifs hébergés par le véhicule (les « payloads » développés par les clients pour leurs besoins propres), des logiciels applicatifs permettant d'assurer un fonctionnement correct du véhicule lui-même (navigation, guidage, gestion des capteurs/actionneurs, etc.), et des logiciels système (moniteur « temps réel », stockage et transfert de données/fichiers, intergiciels pour les traitements distribués, etc.). La « qualité » de ces logiciels est quantifiée (cf. plus loin). Ces logiciels s'exécutent sur des matériels (processeurs, ASIC, etc.) dont la « qualité » est elle aussi quantifiée.

La « qualité » d'un système spatial est principalement liée à la SdF. Bien qu'elles ne soient pas les causes dominantes des échecs ou semi-échecs de missions spatiales, les fautes créées dans les mémoires statiques par ionisation dues aux rayonnements cosmiques (protons, particules alpha, ions lourds, etc.) font l'objet de travaux importants depuis le milieu des années 70 (Karnik *et al.*, 2004)[6]. Une ionisation peut conduire, par exemple, à une modification du contenu d'une position binaire (« bit flip » de mot mémoire ou de registre) ou à un comportement aléatoire d'une porte logique (« glitch », état métastable). On désigne par SEU (single event upset) le phénomène qui conduit à changer la valeur de 1 bit exactement, et par MEU (multiple event upset) celui qui conduit à changer la valeur de plusieurs bits. Bien que les valeurs (données, codes exécutables) contenues dans les mémoires et registres soient protégées par des codes correcteurs et/ou détecteurs (Hamming étendu, par exemple), il subsiste des « failles » de protection, principalement les mémoires-caches et les registres dits « externes », notamment ceux utilisés pour les opérations d'entrée-sortie—ce qui concerne au premier chef les protocoles des sous-systèmes R et les algorithmes fondés sur les échanges de messages utilisés dans les sous-systèmes S.

L'intensité des rayonnements cosmiques est éminemment variable. Elle dépend en particulier des routes suivies et des positions (la « banlieue terrestre » ou l'espace interplanétaire) ainsi que de l'activité solaire. Les taux d'erreurs réels sont donc eux

[6] Des fautes identiques ont été observées dès 1978 avec les mémoires dynamiques au niveau du sol (d'où la nécessité de recourir à des solutions de détection/correction pour, par exemple, les mémoires utilisées dans les véhicules terrestres ou les stimulateurs cardiaques).

aussi éminemment variables, ce qui constitue une cause supplémentaire de variation des délais et débits utiles.

Le « durcissement » des matériels (qualifiés « espace » ou COTS) est une solution qui est employée avec discernement, car conflictuelle avec les contraintes de minimisation de poids et de consommation d'énergie. Pour ces mêmes raisons, on minimise (1) le nombre x de processeurs actifs à tout moment dans un véhicule spatial, (2) le nombre t de processeurs embarqués. En plus des contraintes de coût, la valeur de x est déterminée par les propriétés exigées (cf. plus loin). La valeur de t est déterminée par les caractéristiques des processeurs (fiabilité intrinsèque, etc.) et par la durée prévue de la mission (quelques dizaines de minutes pour un lanceur de satellites, quelques dizaines d'années pour un satellite terrestre à vocation commerciale).

On ne prouver des propriétés de SdF qu'à la condition de spécifier :

- les modèles de défaillances considérés,

- les modèles d'occurrence de ces défaillances,

- le ou les modèles de système/calcul considérés[7].

Dans chacune de ces classes de modèles, il existe une relation d'ordre partiel. Un treillis des modèles de défaillances est donné dans (Powell, 1992). Le modèle le plus restrictif (le plus facile à gérer) est le modèle « arrêt immédiat et définitif ». Le modèle le plus permissif (le plus difficile à gérer) est le modèle « byzantin » (comportements arbitraires dans les domaines des valeurs et du temps). Voir (Lynch, 1996) pour une introduction aux modèles de système/calcul, et (Dolev *et al*., 1987) pour une présentation de 32 modèles partiellement synchrones.

C'est en phase de définition des exigences système (« requirements capture ») dérivées des exigences applicatives que l'on doit spécifier l'ensemble des modèles retenus, ensemble noté M.

2.2. *Services et QoS (propriétés exigées)*

2.2.1. *Communications et réseaux (sous-systèmes R)*

Comme exemples de propriétés exigées, on trouve :

- vivacité (« liveness ») : absence d'interblocage ou de boucles de routage entre les noeuds, absence de congestion (ou congestions contrôlées), etc.

- ponctualité (le « temps réel ») : livraison des messages dans des délais ou/et des gigues connues (en l'absence de surcharges), minimisation d'une fonction de coût en

[7] Des travaux récents permettent d'exprimer ces modèles de façon unifiée, pour les systèmes fonctionnant par rondes (Charron-Bost *et al*., 2006).

présence de surcharge, capacité à distinguer entre congestion et défaillance (de route, de nœuds), etc.,

- confiance : fiabilité (livraisons sans erreurs des messages), disponibilité, authentification, diffusion fiable, etc.,

- désignation adaptative : possibilité de changer ou calculer dynamiquement les noms des points terminaux (« late binding »).

Les protocoles qui existent actuellement pour les réseaux sol-espace sont ceux définis par le « Consultative Committee for Space Data Systems » (CCSDS). Les codes correcteurs d'erreurs (Reed Solomon, notamment) sont utilisés abondamment. Les interactions entre le CCSDS et les communautés réseau (IETF, etc.) se développent, en vue de créer de nouvelles normes. Par exemple, entre le niveau « lien point-à-point » et le niveau « routage », comme entre ce dernier et le niveau « liaison de bout-en-bout », il est proposé que les interactions soient explicites (approches « cross-layer », « bundles »), afin d'optimiser les performances. Pour le routage dans les réseaux espace-espace, le CCSDS a élaboré un ensemble de normes (Space Communication Protocol Standards — Network Protocol) qui sont paramétrables en fonction des missions. Des algorithmes assurant du routage en présence de défaillances (« epidemic routing », DTN, etc.) sont également en cours de normalisation.

Pour le niveau transport, de nombreux travaux concernent TCP. Le protocole d'origine est bien adapté aux réseaux pour lesquels les pertes de messages sont essentiellement dues aux congestions. Lorsque le facteur dominant des pertes de messages est l'existence de défaillances, ou bien lorsque les partitionnements réseau sont fréquents, des variantes de TCP sont nécessaires, telles TCP-DOOR, TCP-RTO, Split TCP (Hanbali *et al.*, 2005), et plus récemment, LTP (Burleigh *et al.*, 2006).

2.2.2. *Traitements (sous-systèmes $S \cup C$)*

Les processus (d'implantation logicielle ou matérielle) embarqués spatial doivent être vérifiés avant emploi. Les coûts de vérification croissant de façon plus que linéaire avec la complexité et/ou la taille des processus, il existe une catégorisation de la « criticité » des processus, comme, par exemple, les SIL en Grande-Bretagne (4 « software integrity levels ») et les 5 niveaux A, B, C, D, E de DO-178B (EOCA, 1992). Les fonctions les plus critiques (SIL 4, niveau A) doivent être assurées par des processus dont la probabilité d'erreur est montrée être au plus égale à 10^{-9}.

Actuellement, le nombre x de processeurs actifs dans un véhicule spatial va de 1 (satellites à bas coût) à 4 (activés pour certaines phases dans le cas des véhicules de transfert d'astronautes ou navettes spatiales)[8]. Le dimensionnement $x = 4$ résulte de la borne $x > 3f$ établie dans (Lamport *et al.*, 1982) pour tolérer au plus f processeurs

[8] Hormis les phases en vol balistique (sondes d'exploration lointaine), pouvant durer plusieurs semaines ou mois, au cours desquelles aucun processeur n'est actif.

défaillants selon le modèle byzantin. L'intérêt de ce modèle est qu'il couvre tous les comportements possibles. Le choix de f = 1 résulte d'une analyse de taux de couverture (sur une durée de l'ordre de 20 minutes, la probabilité d'occurrence d'au moins 2 défaillances byzantines est infinitésimale).

Sauf pour les missions courtes, il existe une réserve (de taille initiale égale à t-x) de processeurs de « rechange », utilisés pour poursuivre une (phase de) mission dans les conditions nominales malgré la défaillance définitive (usure, panne accidentelle) d'un ou plusieurs processeur(s). C'est par exemple le cas pour une phase d'entrée d'un ATV dans l'atmosphère terrestre (de l'ordre de 20 minutes) ou pour un satellite de télécommunications en orbite terrestre (de durée de vie de 20 à 30 ans environ).

La plupart des logiciels applicatifs ou système utilisés actuellement sont à flot de contrôle séquentiel. L'évolution vers le parallélisme et la distribution est inéluctable, notamment pour des raisons de SdF, de réduction des temps de réponse, et d'autonomie.

Un service fondamental dans les applications spatiales est la possibilité de pouvoir télé-(re)charger un logiciel embarqué espace depuis le sol.

Les propriétés exigées, notées P, appartiennent aux 4 classes connues suivantes :

- sûreté logique (« safety ») : atomicité, sérialisabilité, exclusion mutuelle, etc.,

- vivacité (« liveness ») : absence d'interblocage ou quasi-interblocage, terminaison inévitable (« eventual »), etc.,

- ponctualité (le « temps réel ») : tenue d'échéances ou/et de gigues en l'absence de surcharges, maximisation d'une fonction de gain en présence de surcharge, etc.,

- confiance (la SdF) : diffusion fiable, disponibilité, accord approché/exact, etc., en présence de défaillances.

Les deux premières classes réunissent les propriétés dites logiques. Le but des preuves (validation d'une spécification système) est le suivant :

- établir qu'une architecture dotée de certains protocoles/algorithmes possède des propriétés P*, pour des modèles M*,

- démontrer que P* domine P (on prouve qu'existe au moins ce qui est demandé) et que M domine M* (on ne suppose pas plus que ce qui est donné), la domination étant définie d'après les relations d'ordre partiel des classes de modèles et de propriétés.

On examine à présent quelques problèmes majeurs posés par les systèmes spatiaux. La sélection a été faite de façon à mettre en exergue tour à tour le « temps réel », le « distribué », et la « tolérance aux fautes ». Pour faciliter le travail du lecteur, chaque section commence avec une étude de cas. Les cas retenus concernent des missions de la NASA, car ils sont bien documentés. Des problèmes équivalents ont été vécus avec des missions de l'ESA (cf. Ariane 5, vol 501, 1996).

3. Communications et transferts de données sous contraintes temps réel

3.1. *Mars Pathfinder*

Quelques jours après l'arrivée de la sonde Pathfinder et du rover Sojourner sur Mars en juillet 1997, Pathfinder devint silencieux. Le centre de contrôle JPL/NASA (Pasadena) envoya un ordre de redémarrage, et Pathfinder put émettre à nouveau, avant de redevenir silencieux. Ce scénario fut répétitif. A chaque fois, il fallait attendre le lendemain pour redémarrer Pathfinder, ce qui compromettait la collecte de données, et à terme la mission, le nombre de jours opérationnels étant limité (batteries de durées limitées).

Par analyse du système « copie » disponible au sol, le JPL finit par découvrir la cause du problème, présentée comme « une inversion de priorités ». Une tâche A de haute priorité (a) et une tâche C de basse priorité (c) partagent un sémaphore X. L'exécution de C peut être suspendue par des tâches B, B', etc., de priorité moyenne b ($c < b < a$) qui n'utilisent pas X. Dans certaines circonstances, le nombre de réquisitions subies par C était tel que C ne se terminait pas « à temps ». Toutes les 125 ms, une tâche N est activée pour initialiser un nouveau cycle (de 125 ms). Aucune tâche n'est censée être en exécution lorsque N est activée. Lorsque N détectait que C était encore en exécution, N déclenchait un « reset » général du système, Pathfinder se mettant alors en attente des ordres du sol. Le moniteur de tâches utilisé (VxWorks de Wind River) offre l'option « héritage de priorité » (HP), qui permet à une tâche de basse priorité (C ici) qui bloque une tâche de plus haute priorité (A ici) d'acquérir sa priorité (a ici). Le booléen HP avait été mis à « faux ». Si HP avait été mis à « vrai », l'exécution de C n'aurait pu être suspendue par les tâches B, B', etc. Et les « resets » n'auraient pas eu lieu.

3.2. *Nature des problèmes et solutions*

Pathfinder peut être vu comme un élément du sous-système $S \cup C$, mais aussi élément du sous-système R, en tant que nœud mobile (avec Sojourner) du réseau sol-espace, qui a pour autres nœuds les deux orbiteurs de Mars, l'ISS, et les centres au sol de la NASA.

Prima facie, on est en présence d'un scénario typique de réseau DTN : un nœud défaillant interdit les communications entre équipements interconnectés, à savoir les capteurs du rover et les bases de données du segment sol.

3.2.1. *Problèmes et solutions « réseau »*

L'emploi de protocoles de réseaux DTN ou les variantes de TCP pour le spatial n'auraient pas permis d'éviter les problèmes de Pathfinder. Il y avait bien « congestion » *en l'absence de défaillance* (tous les équipements du rover étaient corrects et opérationnels). Dans un tel scénario :

- les mécanismes de contrôle de flux s'activent, et imposent aux sources de réduire leurs débits d'émission,

- le nœud concerné ne s'arrête pas de fonctionner.

C'est le scénario exactement opposé qui fut instancié par Pathfinder : aucun contrôle des sources (les capteurs et les événements déclencheurs des tâches B, B', etc.) n'est prévu/autorisé, et l'arrêt du système embarqué sur le rover est volontaire.

Il s'agit d'une « solution » couramment utilisée dans le spatial. Quand on ne sait pas « ce qui se passe », on bascule en mode dit « safe » : toute activité est stoppée et on attend des ordres de l'extérieur (sol ou autre). Il existe toujours la possibilité que les véritables conditions opérationnelles soient « pires » que ce qui est spécifié (en début de projet) via les modèles M, et dans ce cas le passage en mode « safe » est une transition raisonnable, à condition qu'elle soit exécutée sans violer les propriétés logiques spécifiées via P. Mais il arrive aussi que « ne pas savoir ce qui se passe » en opérationnel est la manifestation d'une faute de conception initiale : on arrête de façon prématurée une analyse devenue trop « compliquée » (car mal conduite). Passer systématiquement en mode « safe » et obéir ensuite aux ordres d'un centre de contrôle peut être dangereux. Par chance, la mission Pathfinder a pu être sauvée. Par contre, c'est exactement cela qui a entraîné la perte du satellite européen SPOT 3.

Il s'agit donc d'un problème *système*. Très précisément, il s'agit d'un problème d'ordonnancement temps réel. Il existe une grande variété d'algorithmes d'ordonnancement de *messages* enfouis dans les protocoles réseau, et utilisés dans les nœuds. Une différence notable avec les problèmes d'ordonnancement de *tâches* posés par les systèmes critiques est l'obligation, pour ces derniers, de garantir la tenue de temps de réponse imposés, y compris et surtout lors des scénarios pires cas.

3.2.2. *Problème et solutions « système »*

Il y avait donc « congestion » *en l'absence de défaillance*. La cause des « resets » est une faute d'ingénierie système. En effet, même avec l'option HP = vrai, il se peut que des tâches violent leurs échéances (de terminaison au plus tard). La seule façon de prouver que ceci ne peut se produire est de conduire une analyse d'ordonnançabilité, de donner les expressions analytiques des bornes supérieures des temps de réponse, et d'établir les conditions de faisabilité « temps réel » pires cas— un système de contraintes liant charges, bornes supérieures des temps de réponse, et échéances[9]. Ce qui ne fut pas fait.

Cette exigence (analyses d'ordonnançabilité) doit être satisfaite pour tout système, protocole, algorithme, fondé sur des réveils (« timers ») et assurant des services critiques. En effet, sans une connaissance exacte des bornes supérieures (de temps de réponse, de latence de détection de défaillance, etc.), comment peut-on estimer les « bonnes » valeurs de réveils ? De fait, cette exigence est l'un des obstacles majeurs à l'emploi du modèle de système/calcul synchrone pur dans les

[9] Cf. théorie de l'ordonnancement (et les actes du Real-Time Systems Symposium, IEEE CS).

systèmes distribués en général, dans les systèmes critiques en particulier. Calculer un wcet (« worst-case execution time ») d'un programme/logiciel s'exécutant de façon isolée est relativement aisé. Il est autrement plus complexe d'établir, pour chaque programme/logiciel qui s'exécute de façon multiplexée (en concurrence avec d'autres programmes/logiciels) sur une architecture et des ressources partagées, une borne supérieure des temps de séjour en file(s) d'attente (dans un réseau et/ou dans un système distribué (problèmes combinatoires NP-durs)).

Pour mener ce type d'analyses, il est nécessaire de spécifier au préalable le ou les algorithmes d'ordonnancement choisi(s). A cause de leur simplicité, ce sont quasiment toujours des algorithmes à priorités fixes qui sont choisis pour le spatial, variantes de l'algorithme Highest-Priority-First (HPF) décrit initialement dans (Liu *et al.*, 1973). Malheureusement, contrairement à Earliest-Deadline-First (EDF), présenté dans le même article, ces algorithmes ignorent les échéances des tâches en attente d'exécution. Il existe donc de nombreux scénarios faisables avec EDF qui ne le sont pas avec HPF (Buttazzo, 2005).

4. Atomicité des transactions en présence de réseau partitionné

4.1. *Mars Rover Spirit*

Dès après le lancement de la sonde et du rover Spirit (juin 2003), le JPL (Jet Propulsion Laboratory, NASA) décida de télécharger une nouvelle version du code exécutable, des problèmes de saturation des zones mémoire (notées Z) allouées au stockage des données collectées par les caméras ayant été détectés. Ces données constituent des fichiers qui sont transmis vers le sol lorsque les communications espace-sol sont possibles.

Mais cette opération ne donna pas les effets escomptés. A compter du 21 janvier 2004, le rover Spirit devint silencieux, de façon permanente (à la différence de Mars Pathfinder). Les analyses conduites au centre de contrôle permirent de découvrir que les répertoires (« directories ») des fichiers (notés D) étaient restés ceux de la version devenue obsolète (ceux correspondant à l'image initiale de Z), ce qui conduisait à des débordements de Z. En cas de débordement, une exception est levée par le moniteur VxWorks. A la différence du choix fait pour Pathfinder, ceci ne conduisait pas à un arrêt (« reset » et attente d'ordre du sol) du système, mais à une réinitialisation (« reboot ») automatique, couronnée de succès de façon fugitive, les débordements de Z survenant quasiment immédiatement.

Un utilitaire U fut alors téléchargé, dans le but de remplacer les anciens répertoires par les nouveaux, afin de rétablir la cohérence entre D et Z. Mais le téléchargement fut incomplet : un seul des deux modules constituant U fut transmis avec succès, pour cause de partitionnement temporaire du réseau sol-espace. Et les débordements continuèrent … jusqu'à ce qu'un ingénieur découvrit qu'il fallait télécharger le second module de U. Mais il fallut aussi « nettoyer » la mémoire, par

le biais de commandes manipulant directement les adresses mémoire (!), afin de supprimer de l'ordre de 1.000 fichiers inutiles.

4.2. *Nature des problèmes et solutions*

Le scénario Spirit révèle l'existence de dimensionnements erronés des zones mémoires, des fautes d'ingénierie système. Mais ce qui nous concerne plus directement est la question suivante : si un protocole réseau capable de tolérer un partitionnement réseau avait été utilisé, aurait-il été possible d'éviter les incohérences/débordements causés par la réception en deux opérations séparées des deux modules de U ?

4.2.1. *Problème et solutions réseaux*

La réponse est oui et non. Oui, un protocole de bout-en-bout (variante de TCP pour le spatial) aurait permis d'éviter de perdre des jours de mission (et de compter sur la sagacité d'un ingénieur pour terminer la transmission de U). Dès disparition du partitionnement réseau, le second module de U aurait été transmis à Spirit. Mais ceci n'aurait pas été suffisant (d'où le « non »). En effet, l'opération désirée doit, pour être correcte, apparaître comme indivisible au reste du système : on remplace de façon atomique les anciens répertoires et les fichiers créant les débordements. Il s'agit donc d'une *transaction atomique* (Bernstein *et al*., 1987).

En cas de partitionnement réseau, le transfert de bout-en-bout s'exécute comme deux sessions différentes (2 identités différentes sont utilisées pour la liaison). Il revient aux processus impliqués (source, destinataire) de « savoir » que les 2 actions/transferts de modules de U constituent une opération (transaction) qui doit rester atomique : soit on n'exécute aucune de ces actions, soit on les exécute « ensemble », et ceci sans bloquer.

Il s'agit donc du problème de terminaison atomique non bloquante, noté TANB. Bien que n'étant pas distribuée dans le cas de Spirit, la transaction ne s'est pas terminée de façon atomique … parce que le système embarqué sur Spirit n'est pas doté d'un algorithme approprié (cf. plus loin).

4.2.2. *Problème et solutions système*

Une transaction X est un ensemble de n actions, que l'on appellera processus. Dans le cas général, X s'exécute sur plusieurs processeurs. Les processus effectuent des opérations d'écriture sur des variables rémanentes, qui sont lues (et écrites) par d'autres transactions. Les écritures sont effectuées sur des copies, non visibles, et peuvent donc être annulées. Pour terminer X, il est nécessaire de rendre publiques et irréversibles toutes les écritures, ou bien aucune d'entre elles. Ainsi, pour chaque écriture, le processus responsable doit indiquer s'il vote « commit » (la valeur écrite peut être rendue publique) ou bien « abort » (l'écriture est impossible). Il se peut

qu'un processus soit lui-même défaillant (silencieux) ou bien que des messages échangés entre processus de X se perdent ou circulent trop lentement.

Afin de garantir des propriétés telles la causalité ou la sérialisabilité des exécutions, il est nécessaire d'assurer l'atomicité de terminaison et la vivacité en présence de défaillances. Il s'agit d'un problème non trivial, et il existe des résultats d'impossibilité. Le problème TANB est traditionnellement défini pour des communications fiables et des défaillances de type arrêt immédiat pour les processeurs. Initialement, chaque processus (d'une transaction X) vote 0 (« abort ») ou 1 (« commit ») d'après ses conditions locales, et doit décider de l'issue globale de X, par exécution d'un algorithme de TANB. Un algorithme résout TANB pour f défaillances, $0 < f < n$, si lors de chaque exécution de cet algorithme en présence de f défaillances au plus, on a les propriétés suivantes :

- Terminaison : tout processus correct finit par décider
- Accord : tous les processus décident identiquement
- Validité 1 : si au moins 1 processus vote 0 initialement, alors la seule décision possible est « abort X »
- Validité 2 : si tous les processus votent 1 initialement, et s'il ne se produit aucune défaillance, alors la seule décision possible est « commit X ».

Validité 2 est essentielle. Sans elle, une solution triviale consiste à décider « abort X » systématiquement. Mais c'est à cause de Validité 2 que le problème TANB est non trivial.

Lorsque les conditions sont « mauvaises » (défaillances non réparées ou récurrentes), on n'exige que des propriétés de sûreté logique (« safety ») : aucun progrès mais pas de perte d'atomicité. Lorsque les conditions sont ou redeviennent « bonnes » (délais dans les plages de valeurs espérées, pas de défaillance), on exige la terminaison, en plus de l'atomicité.

Le problème de la terminaison atomique en présence de défaillances est examiné depuis le début des années 80 – voir (Bernstein *et al.*, 1987). Les premiers algorithmes tels 2-phase locking ou 3-phase locking (Skeen, 1981) sont bloquants, c'est-à-dire qu'ils ne terminent pas en présence de certains scénarios de défaillances.

Depuis, de nombreuses publications ont été consacrées au problème TANB. Cependant, même les plus récentes (par exemple, Gray *et al.*, 2006) ne résolvent pas ce problème de façon satisfaisante. Les solutions en présence sont évaluées et comparées presqu'exclusivement selon des mesures de complexité (messages, temps) en moyenne, en pire cas, en meilleur cas. Mais le critère fondamental pour TANB est le taux de succès : lorsque les conditions permettent de décider « commit », quelle est la proportion de terminaisons avec « commit » (par rapport aux terminaisons avec « abort ») qui est garantie par un algorithme ?

Il est en effet très facile de résoudre TANB en décidant « abort » systématiquement. Et point n'est besoin d'utiliser un algorithme complexe pour atteindre ce résultat inintéressant. A ce jour, cette question reste ouverte.

5. Accord approché et consensus en présence de défaillances

L'étude de cas DART met en évidence l'utilité de recourir à un service d'accord capable de tolérer des défaillances dans un sous-système S $\cup$ C. Lorsque c'est possible ou indispensable, le service d'accord est distribué. Dans le cas contraire, un tel service doit être diversifié.

5.1. *DART*

Le 15 avril 2005 fut mis en orbite terrestre par la NASA (Vanderberg, Californie) le véhicule spatial DART (Demonstration of Autonomous Rendezvous Technology), chargé d'effectuer plusieurs manœuvres en proximité rapprochée du satellite MUBLCOM (Multiple Paths, Beyond-Line-of-Sight Communications), lancé en 1999. Le but de DART était de démontrer la faisabilité d'opérations totalement autonomes. La mission devait durer 24 heures. Au bout de 11 heures, DART détecta qu'il avait presque épuisé ses réserves de carburant, et il décida d'entamer la série de manœuvres d'éloignement. En fait, 3 minutes et 49 secondes avant d'initialiser cette série, DART était entré en collision avec MUBLCOM, ce qu'il n'avait pas détecté (il réussit néanmoins à s'éloigner de MUBLCOM après la collision).

Le positionnement de DART par rapport à MUBLCOM était assuré par un capteur AVGS (Advanced Video Guidance Sensor) et 3 récepteurs GPS (dont 2 sur DART). Les signaux laser émis par l'AVGS étaient renvoyés par des réflecteurs montés sur MUBLCOM, pour calculer distances et attitudes relatives. Par combinaison des données (fonctionnellement redondantes) en provenance de l'AVGS et des récepteurs GPS, le logiciel de navigation embarqué sur DART devait garantir des calculs précis et exacts, en particulier ceux donnant les distances relatives DART-MUBLCOM.

Ce ne fut pas le cas. Vitesse d'approche trop élevée, distance estimée supérieure à la valeur réelle : bien que le système d'évitement de collision de DART se soit mis en route 1 minute et 23 secondes avant l'impact, ce dernier ne put être évité.

Les causes des deux dysfonctionnements furent identifiées par analyse des informations de télémétrie.

La consommation excessive de carburant, et donc la fin prématurée de la mission (qui serait survenue même en l'absence de collision), sont dues à des défaillances par valeurs (informations d'entrée et calculs erronés) répétées de certains capteurs et du

logiciel de navigation. A titre d'exemple, les mesures de vitesse assurées par récepteur GPS primaire et son logiciel furent constamment « biaisées » de 0,6 m/s. Ces défaillances causèrent un nombre anormalement élevé de corrections de trajectoire, consommatrices de carburant.

La collision est elle aussi due aux défaillances par valeurs dans C (capteurs) et S (logiciel de navigation et logiciel d'évitement de collision). En fait, la collision est le résultat d'une faute élémentaire de conception système, en violation d'une règle essentielle lorsqu'il s'agit de systèmes critiques, à savoir qu'il ne doit pas exister de possibilité de défaillance en mode commun. Lorsqu'une analyse conclut à cette possibilité, il est requis de redonder de façon diversifiée (capteurs et logiciels dans le cas de DART). Le logiciel d'évitement de collision et le logiciel de navigation avaient la *même* source d'informations (erronées) de positionnement – le « mode commun ». C'est ainsi qu'au moment de la collision, DART avait une vitesse de 1,5 m/s, alors que son logiciel de navigation indiquait (1) que DART était en train de s'éloigner de MUBLCOM à une vitesse de 0,3 m/s, (2) une distance DART-MUBLCOM de 130 mètres.

5.2. *Nature des problèmes et solutions*

Dans le cas de DART, il s'agit d'un rendez-vous à 2 entités, dont la réussite reposait entièrement sur le fonctionnement correct d'une seule entité active (n = 1), l'autre étant passive. Un rendez-vous spatial autonome pose en particulier le problème de l'accord approché (cf. plus loin). Les conditions de faisabilité sont connues (n > f en présence de au plus f défaillances). Dans le cas de DART, avec le scénario réel, on avait f = 1. Mais l'hypothèse non validée posée par les responsables de la mission était f = 0.

Dans les systèmes autonomes ou/et critiques, soit les décisions sont centralisées, et l'on prouve que l'entité centralisatrice a une probabilité de défaillance inférieure à un seuil imposé, soit les décisions sont prises de façon « collégiale » (à plusieurs), par le biais d'un algorithme de décision D (accord approché ou accord exact), et l'on prouve qu'un tel algorithme peut tolérer jusqu'à f défaillances concomitantes correspondant au modèle le plus « permissif » retenu.

Dans le cas de DART, on peut imaginer deux types de solutions correctes.

- *Seul DART est actif*

Il est nécessaire de diversifier les capteurs mais aussi les logiciels (ni le calculateur embarqué (S), ni les communications (R) — inexistantes, ne sont impliqués dans les dysfonctionnements). A intervalles réguliers, chacune des instances diversifiées (du logiciel d'évitement de collision, par exemple) effectue un calcul à partir des informations lues sur les capteurs, par le biais d'un algorithme d'accord approché (AA). Les algorithmes AA sont soit coopératifs (chaque instance soumet son calcul aux autres), soit unilatéraux (pas d'échanges entre instances).

L'intérêt d'un algorithme AA est qu'il permet de masquer ou de détecter l'existence de calculs erronés.

- *DART et MUBLCOM sont actifs*

Les deux entités impliquées coopèrent pour réussir le rendez-vous autonome. La plupart des véhicules spatiaux étant équipés d'un utilitaire permettant le téléchargement de logiciels, on peut envisager ceci : (1) téléchargement préalable sur MUBLCOM d'un algorithme AA, et activation des antennes, (2) en phase d'approche, exécution périodique de AA par MUBLCOM et DART. Dans les mêmes conditions, MUBLCOM aurait été en mesure de détecter le comportement anormal de DART, et de lui faire activer *à temps* son système d'évitement de collision. A condition toutefois que les communications soient « suffisamment » fiables (problèmes pour R).

L'intérêt principal des algorithmes distribués tolérants aux défaillances est leur simplicité, et donc celle de leurs implantations en logiciel. Par comparaison, les logiciels de navigation-guidage sont très complexes. Il est donc bien plus facile de valider et vérifier un algorithme d'accord approché (ou exact) et son implantation que la conception et l'implantation d'un logiciel de navigation, tel celui embarqué sur DART. De plus, le travail de conception-validation-vérification n'est exécuté qu'une fois pour un algorithme système générique, alors que ce travail doit être effectué pour tout nouveau logiciel applicatif, ou pour toute nouvelle version d'un logiciel applicatif existant.

Dans les systèmes conçus d'emblée pour fonctionner de façon autonome (« formation flying », constellations et/ou flottilles de satellites ou véhicules spatiaux, etc.), on dispose de n ($n > 2$) processus distribués sur les véhicules spatiaux. De façon récurrente, ces processus doivent obtenir un accord approché (AA) ou un accord exact—connu sous le nom de consensus (CS).

5.2.1. *Accord approché*

Le problème AA est celui de la distribution des données générées par k sources à n processus distincts, en présence de f défaillances au plus, de telle sorte que les processus (1) calculent à peu près les mêmes valeurs d'entrée depuis les données reçues, (2) produisent à peu près les mêmes valeurs de sortie. Ce problème se pose avec des sources continues (« sensor fusion ») ou discrétisées (les horloges, à partir desquelles on entretient une base de temps global distribué).

Bien qu'un algorithme de diffusion atomique (Lynch, 1996) permette de résoudre des problèmes plus durs que AA, son emploi est souvent soit impossible (il faut « équiper » les sources d'un tel algorithme), soit trop coûteux en messages et en temps (il faut dérouler au moins f+1 rondes), car fournissant des types d'accord inutilement précis pour les problèmes posés.

Un algorithme AA doit contourner la difficulté qui tient aux « distorsions » crées par les défaillances (Dolev et al., 1986, Marzullo, 1990). Une technique efficace

consiste à éliminer les t valeurs les plus petites et les t valeurs les plus grandes, puis à retenir la moyenne ou la médiane des valeurs restantes (on peut aussi travailler sur des intervalles plutôt que sur des valeurs).

Dans le cas de défaillances bénignes, on a t = f/2 et n > 2f. Dans le cas de défaillances byzantines, on a t = f et n > 3f.

On prouve que pour tout couple quelconque de processus, la distance entre les valeurs ou les intervalles calculés ne peut être supérieure à une grandeur calculable, qui dépend en particulier de k, n et de f.

5.2.2. *Consensus*

Le problème du consensus est traditionnellement défini pour des communications fiables et des défaillances de type arrêt immédiat pour les processeurs (Lynch, 1996). Chaque processus propose une valeur initiale choisie librement. Tout processus non défaillant doit choisir de façon irrévocable l'une des valeurs proposées comme valeur de décision (finale). On a consensus si tous les processus qui décident choisissent la même valeur.

Un algorithme CS résout consensus pour f défaillances, 0 < f < n, si lors de chaque exécution de cet algorithme en présence de f défaillances au plus, on a les propriétés suivantes :

- Terminaison : tout processus correct finit par décider
- Accord : tous les processus décident identiquement
- Validité : la valeur de décision est l'une des valeurs proposées.

La propriété de terminaison inévitable (« eventual ») est remplacée par la propriété de terminaison ponctuelle suivante dans le consensus « temps réel » :

- Terminaison : tout processus correct décide en au plus Δ unités de temps après début d'exécution de CS.

Le consensus est posé par des besoins applicatifs comme, par exemple, élection de leader, changement de mode opérationnel, reconfiguration (par exemple, exclusion ou non exclusion d'un véhicule), démarrage coordonné d'une opération applicative distribuée (par exemple, modification des positions dans une formation).

Consensus (accord exact) est plus dur que l'accord approché. Il existe de nombreux résultats d'optimalité pour consensus, dans divers modèles de système/calcul. L'un des résultats fondamentaux (Fischer *et al.*, 1985) est l'impossibilité de consensus (déterministe) en modèle asynchrone pur (on ne peut supposer l'existence de bornes sur les délais de communication/calcul), en présence de défaillance – même si (1) les communications sont fiables, (2) seulement 1 processus défaille (arrêt immédiat et définitif, le modèle le plus restrictif connu).

Pour « contourner » ce résultat, il est nécessaire soit de définir des conditions suffisantes temporaires vérifiées pour certaines étapes d'exécution de CS (Charron-

Bost *et al.*, 2006), soit d'enrichir le modèle asynchrone pur. Les détecteurs de défaillances imparfaits (Chandra *et al.*, 1996) sont une formalisation axiomatique d'un tel enrichissement bien explorée depuis une douzaine d'années.

Dans (Hermant *et al.*, 2002), on montre (1) comment l'on ramène le problème d'implantation des détecteurs de défaillances à un problème d'ordonnancement temps réel de messages dans un réseau, (2) comment construire des détecteurs de défaillances rapides (nécessaire pour le temps réel), et l'on donne un algorithme de CS rapide. Le tout a été implanté par EADS Astrium en 2003[10] (Honvault *et al.*, 2005).

6. Conclusions

Les applications spatiales posent un certain nombre de défis, notamment aux deux communautés « réseau » et « système ». Des analyses de problèmes vécus lors de missions spatiales ont été présentées afin d'illustrer quelques problèmes génériques ainsi que leurs solutions. Ce qui est brièvement exposé dans cet article ne représente qu'une fraction de l'état de l'art pertinent pour relever ces défis. D'autres domaines applicatifs (« critiques ») partagent un certain nombre de points communs avec le spatial. Il est donc permis de penser qu'une factorisation des travaux et des connaissances nécessaires est non seulement possible mais souhaitable. Puisse cet article y contribuer.

7. Bibliographie

Akyildiz I., Akan Ö., Chen C., Fang J., Su W., « The State of the Art in Interplanetary Internet », *IEEE Communications Magazine*, juillet 2004, p. 108-118.

Bernstein P.A., Hadzilacos V., Goodman N., *Concurrency Control and Recovery in Database Systems*, Addison-Wesley Publishing Company, 1987, 320 p.

Bowen J.P., Hinchey M.G., « Ten Commandments of Formal Methods … Ten Years Later », *IEEE Computer*, janvier 2006, p. 41-48.

Burleigh S., Ramadas M., Farrell S., « Licklider Transmission Protocol – Motivation », Internet draft, draft-irtf-dtnrg-ltp-motivation-02.txt, mars 2006.

Buttazzo G., « Rate Monotonic vs. EDF: Judgement Day », *Real-Time Systems Journal (Kluwer)*, vol. 29, n°1, janvier 2005, p. 5-26.

CCSDS, http://www.ccsds.org

Cerf et al., « Interplanetary Internet (IPN): Architectural Definition », Internet draft, draft-irtf-ipnrg-arch-00.txt, mai 2001.

[10] Contrat A3M, 2001-2003, financé par l'Agence Spatiale Européenne (ESTEC).

Chandra T., Toueg S., « Unreliable Failure Detectors for Reliable Distributed Systems », *Journal of the ACM*, vol. 43, n°2, 1996, p. 225-267.

Charron-Bost B., Schiper A., « The Heard-Of Model: Unifying all Benign Failures », Technical Report, EPFL (CH), juillet 2006, 37 p.

Dolev D., Dwork C., Stockmeyer L., « On the Minimal Synchronism Needed for Distributed Consensus », *Journal of the ACM*, vol. 34, n°1, 1987, p. 77-97.

Dolev D., Lynch N.A., Pinter S.S., Stark E.W., Weihl W.E., « Reaching Approximate Agreement in the Presence of Faults », *Journal of the ACM*, vol. 33, n°3, juillet 1986, p. 499-516.

EOCA, *Software Considerations in Airborne Systems and Equipment Certification*, DO-178B, Requirements and Technical Concepts for Aviation, European Organization for Civil Aviation Equipment, 1992.

Fischer M.J., Lynch N.A., Paterson M.S., « Impossibility of Distributed Consensus with One Faulty Process », *Journal of the ACM*, vol. 32, n°2, avril 1985, p. 374-382.

Gray J., Lamport L., « Consensus on Transaction Commit », *ACM Transactions on Database Systems*, vol. 31, n°1, mars 2006, p. 133-160.

Hanbali A.A., Altman E., Nain P., « A Survey of TCP over Ad Hoc Networks », *IEEE Communications Surveys & Tutorials*, vol. 7, n°3, 2005, p. 22-36.

Hermant J.-F., Le Lann G., « Fast Asynchronous Uniform Consensus in Real-Time Distributed Systems », *IEEE Transactions on Computers*, vol. 51, n°8, août 2002, p. 931-944.

Hoare T., « The Verifying Compiler: A Grand Challenge for Computing Research », *Journal of the ACM*, vol. 50, n°1, janvier 2003, p. 63-69.

Honvault C., Le Roy M., Gula P., Fabre J.C., Le Lann G., Bornschlegl E., « Novel Generic Middleware Building Blocks for Dependable Modular Avionics Systems », *Proceedings of the 5th European Conference on Dependable Computing (EDCC-5), Lecture Notes in Computer Science (Springer)*, n°3463, avril 2005, p. 140-153.

Karnik T., Hazucha P., Patel J., « Characterization of Soft Errors Caused by Single Event Upsets in CMOS Processes », *IEEE Transactions on Dependable and Secure Computing*, vol. 1, n°2, avril-juin 2004, p. 128-143.

Lamport L., Shostak R., Pease M., « The Byzantine Generals Problem », *ACM Transactions on Programming Languages and Systems*, vol. 4, n°3, juillet 1982, p. 382-401.

Le Lann G., « Proof-Based System Engineering and Embedded Systems », *Proceedings of the European School on Embedded Systems, Veldhoven (NL), Lecture Notes in Computer Science (Springer)*, n°1494, octobre 1998, p. 208-248.

Liu C.L., Layland J.W., « Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment », *Journal of the ACM*, vol. 20, n°1, janvier 1973, p. 46-61.

Lynch N.A., *Distributed Algorithms*, Morgan Kaufmann, mars 1996, 907 p.

Marzullo K.A., « Tolerating Failures of Continuous-Valued Sensors », *ACM Transactions on Computer Systems*, vol. 8, n°4, 1990, p. 284-304.

Powell D., « Failure Mode Assumptions and Assumption Coverage », *Proceedings of the, the 22nd IEEE International Symposium on Fault-Tolerant Computing (FTCS 22)*, juillet 1992, p. 386-395.

Skeen D., « Non Blocking Commit Protocols », *Proceedings of the SIGMOD International Conference on Data Management, Mai 2001*, p. 133-142