Etude de la dégradation progressive dans les systèmes repartis

Catherine Bellon

THESE

présentée à

Institut National Polytechnique de Grenoble

pour obtenir le grade de

DOCTEUR DE 3ème CYCLE
Génie Informatique

Catherine BELLON

ETUDE DE LA DEGRADATION PROGRESSIVE DANS LES SYSTEMES REPARTIS.

Thèse soutenue le 14 septembre 1977 devant la Commission d'Examen :

Président : L. BOLLIET

Examinateurs : R. BEAUFILS
E.J. Mc CLUSKEY
D. HARDY
G. LA ROSA
G. SAUCIER

THESE

présentée à

Institut National Polytechnique de Grenoble

pour obtenir le grade de

DOCTEUR DE 3ème CYCLE
Génie Informatique

Catherine BELLON

ETUDE DE LA DEGRADATION PROGRESSIVE DANS LES SYSTEMES REPARTIS.

Thèse soutenue le 14 septembre 1977 devant la Commission d'Examen :

Président : L. BOLLIET

Examinateurs : R. BEAUFILS
E.J. Mc CLUSKEY
D. HARDY
G. LA ROSA
G. SAUCIER

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Monsieur Philippe TRAYNARD : Président

Monsieur Pierre-Jean LAURENT : Vice Président

PROFESSEURS TITULAIRES

MM.	BENOIT Jean	Radioélectricité
	BESSON Jean	Electrochimie
	BLOCH Daniel	Physique du solide
	BONNETAIN Lucien	Chimie minérale
	BONNIER Etienne	Electrochimie et électrométallurgie
	BOUDOURIS Georges	Radioélectricité
	BRISSONNEAU Pierre	Physique du solide
	BUYLE-BODIN Maurice	Electronique
	COUMES André	Radioélectricité
	DURAND Francis	Métallurgie
	FELICI Noël	Electrostatique
	FOULARD Claude	Automatique
	LESPINARD Georges	Mécanique
	MOREAU René	Mécanique
	PARIAUD Jean-Charles	Chimie-Physique
	PAUTHENET René	Physique du solide
	PERRET René	Servomécanismes
	POLOUJADOFF Michel	Electrotechnique
	SILBER Robert	Mécanique des fluides

PROFESSEUR ASSOCIE

M.	ROUXEL Roland	Automatique

PROFESSEURS SANS CHAIRE

MM.	BLIMAN Samuel	Electronique
	BOUVARD Maurice	Génie mécanique
	COHEN Joseph	Electrotechnique
	LACOUME Jean-Louis	Géophysique
	LANCIA Roland	Electronique
	ROBERT François	Analyse Numérique
	VEILLON Gérard	Informatique fondamentale et appliquée
	ZADWORNY François	Electronique

MAITRES DE CONFERENCES

MM.	ANCEAU François	Mathématiques appliquées
	CHARTIER Germain	Electronique
	GUYOT Pierre	Chimie minérale
	IVANES Marcel	Electrotechnique
	JOUBERT Jean-Claude	Physique du solide
	MORET Roger	Electrotechnique nucléaire
	PIERRARD Jean-Marie	Mécanique
	SABONNADIERE Jean-Claude	Informatique fondamentale et appliquée
Mme.	SAUCIER Gabrièle	Informatique fondamentale et appliquée

MAITRE DE CONFERENCES ASSOCIE

M.	LANDAU Ioan	Automatique

CHERCHEURS DU C.N.R.S. (Directeur et Maîtres de Recherche)

MM.	FRUCHART Robert	Directeur de Recherche
	ANSARA Ibrahim	Maître de Recherche
	CARRE René	Maître de Recherche
	DRIOLE Jean	Maître de Recherche
	MATHIEU Jean-Claude	Maître de Recherche
	MUNIER Jacques	Maître de Recherche

Je voudrais exprimer toute ma reconnaissance à

Madame G. SAUCIER, Maître de Conférences à l'ENSIMAG, qui m'a accueillie dans son équipe, m'a initiée à la recherche et a dirigé mon travail avec attention et amitié;

Je remercie vivement

Monsieur L. BOLLIET, Professeur à l'Institut Universitaire de Technologie de Grenoble, qui m'a fait l'honneur d'accepter de présider le jury de cette thèse;

Je tiens à remercier,

Monsieur R. BEAUFILS, Professeur à l'Université Paul Sabatier de Toulouse,

Monsieur D. HARDY, Chef du département RCI.ESC au CNET-LANNION,

Monsieur G. LAROSA, Chef de groupe à la DRME,

Monsieur Mc CLUSKEY, Professeur à l'Université de Stanford,

qui ont bien voulu accepter de faire partie de ce jury.

Je voudrais remercier aussi mes camarades de l'équipe "Conception et Sécurité des Systèmes Logiques", en particulier Mlle C. ROBACH et Mr. P. CASPI, ainsi que Messieurs G. MAZARE, Y. DESWARTES de la CII, Messieurs A. PRIGENT et C. KUBIAK du CNET-LANNION, qui, par de nombreuses discussions, ont contribué à ce travail.

Enfin, je remercie ceux et celles qui ont travaillé à la frappe et au tirage de cette thèse.

-o-o-o-o-

S O M M A I R E

CHAPITRE V : APPLICATION : LE CENTRAL TELEPHONIQUE E10

I - Evaluation de la puissance moyenne de E10

II - Détection en cours de fonctionnement : le GUR

INTRODUCTION

L'un des critères qui conduisent au choix d'une architecture répartie est la tolérance aux pannes du matériel.

On dit qu'un système tolère les pannes s'il continue à assurer ses fonctions après l'apparition de pannes. La tolérance aux pannes peut être assurée :

- Soit par le masquage des erreurs, assuré par redondance massive (TMR, NMR) ou sélective (codes correcteurs). Cette méthode convient particulièrement aux architectures classiques mais se révèle coûteuse en matériel et/ou en temps d'étude.

- Soit par l'utilisation de rechanges : après détection d'une panne dans un composant, un composant semblable (ou un ensemble de composants pouvant réaliser les mêmes fonctions) est mis en service. Cette méthode peut s'appliquer indifféremment aux architectures classiques et aux architectures réparties; dans le deuxième cas, le composant de rechange sera généralement un module. Cette méthode est coûteuse en matériel : matériel de rechange utilisé rarement.

- Soit par la possibilité de dégradation progressive. Cette méthode est la moins coûteuse et s'appliquera surtout aux architectures réparties; mais elle implique une diminution des performances du système. C'est à cette méthode que nous nous intéressons dans cette étude.

Un système admet une dégradation progressive en présence de pannes lorsqu'il peut être automatiquement reconfiguré, logiciellement et matériellement après la détection d'une panne matérielle.

Dans un système réparti, la reconfiguration matérielle consiste à supprimer l'unité défectueuse; la reconfiguration logicielle consiste à adapter le logiciel à la nouvelle structure matérielle et à assurer la répartition des tâches sur les unités restantes. Le système peut donc continuer à fonctionner (à assurer toutes ses fonctions) quoique avec des performances réduites.

L'étude de la dégradation progressive peut donc être abordée sous deux aspects :

- évaluation de la puissance dégradée
- étude de la détection en cours de fonctionnement, nécessaire pour assurer la reconfiguration du système.

Dans le chapitre I, les composantes de la sûreté de fonctionnement sont définies; l'influence de l'adjonction de matériel supplémentaire en vue d'assurer la détection des erreurs est soulignée.

Dans le chapitre II, les définitions classiques de la fiabilité et de la disponibilité sont étendues pour permettre l'évaluation d'un système à dégradation progressive. Diverses configurations sont évaluées.

Dans le chapitre III, une modélisation des processus est proposée, en vue de définir les politiques de détection en cours de fonctionnement. La détection continue et la contamination d'un processus par des erreurs sur ses variables d'entrées sont proposées.

Au chapitre IV, la détection discontinue est étudiée et appliquée à un système multiprocesseur.

Au chapitre V, une application est présentée : le centre téléphonique E10 et son interface avec les lignes MIC : le GUR.

CHAPITRE I

SURETE DE FONCTIONNEMENT D'UN SYSTEME.

I - INTRODUCTION

La "Sûreté de fonctionnement" est une grandeur vectorielle complexe, à plusieurs composantes (fiabilité, disponibilité, sécurité, crédibilité, maintenabilité, réparabilité). Suivant le type du système (automatique, informatique), et son utilisation, l'importance que l'on attache à chaque composante varie. On ne peut donc pas donner une expression générale d'évaluation de la sûreté de fonctionnement ; par contre l'évaluation de ses composantes a un intérêt général.

Ces composantes ont été définies de diverses façons. En (1), les définitions sont basées sur la distinction entre panne bénignes et pannes malignes ; en (2), les composantes sont définies à partir du taux d'échec ; en (3), une distinction entre les méthodes logicielles et les méthodes matérielles est faite. Pour notre part, nous basons nos définitions sur les états d'un modèle général d'évolution .

On définit une panne comme une défaillance du matériel. Une erreur est la manifestation d'une panne : ayant appliqué à l'unité en panne un vecteur (ou une séquence) d'entrée qui est un vecteur (ou une séquence) de test de cette panne, on obtient une erreur, c'est à dire une sortie différente de celle définie par les spécifications du système.

II - MODELE GENERAL DE LA SURETE DE FONCTIONNEMENT.

Un système complexe peut prendre un grand nombre d'états ; suivant l'étude faite, on réalise une partition différente de ces états.

II - 1. ETATS DU SYSTEME

Dans cette étude, on distingue d'abord six états ; cette partition sera affinée par la suite.

Etat Qc : le système fonctionne correctement ; il peut comporter une panne, mais cette panne n'a pas encore provoqué d'erreurs (l'erreur est latente).

Etat Qe : il existe une panne (simple ou multiple) dans le système, qui a provoqué des erreurs ; ces erreurs n'ont pas été détectées ; le système continue à fonctionner, mais de façon erronée.

Etat Qm : il existe une panne dans le système, qui a été détectée, soit en cours de fonctionnement (correct ou incorrect) par les erreurs qu'elle a provoquées, soit par la maintenance préventive. Dans cet état, la maintenance curative du matériel est effectuée : une panne matérielle est cherchée, localisée s'il y a lieu, puis la réparation du matériel est effectuée, soit par intervention humaine, soit par reconfiguration automatique du matériel.

Etat Ql : les conséquences de la panne, au niveau logiciel, sont recherchées; les données et les programmes sont remis en état ou reconfigurés.

Etat Qa : cet état est un état arrêt : la localisation ou la réparation d'une panne et de ses conséquences, s'est révélée impossible à effectuer en respectant les contraintes de fonctionnement imposées.

Etat Qp : cet état est l'état de maintenance préventive. La maintenance préventive consiste en un test du matériel et du logiciel, et à la préparation des informations nécessaires à la réparation du logiciel (création de points de reprise...);cela impose l'arrêt du fonctionnement normal du système (copie de fichiers importants...).

Le diagramme des états est le suivant :

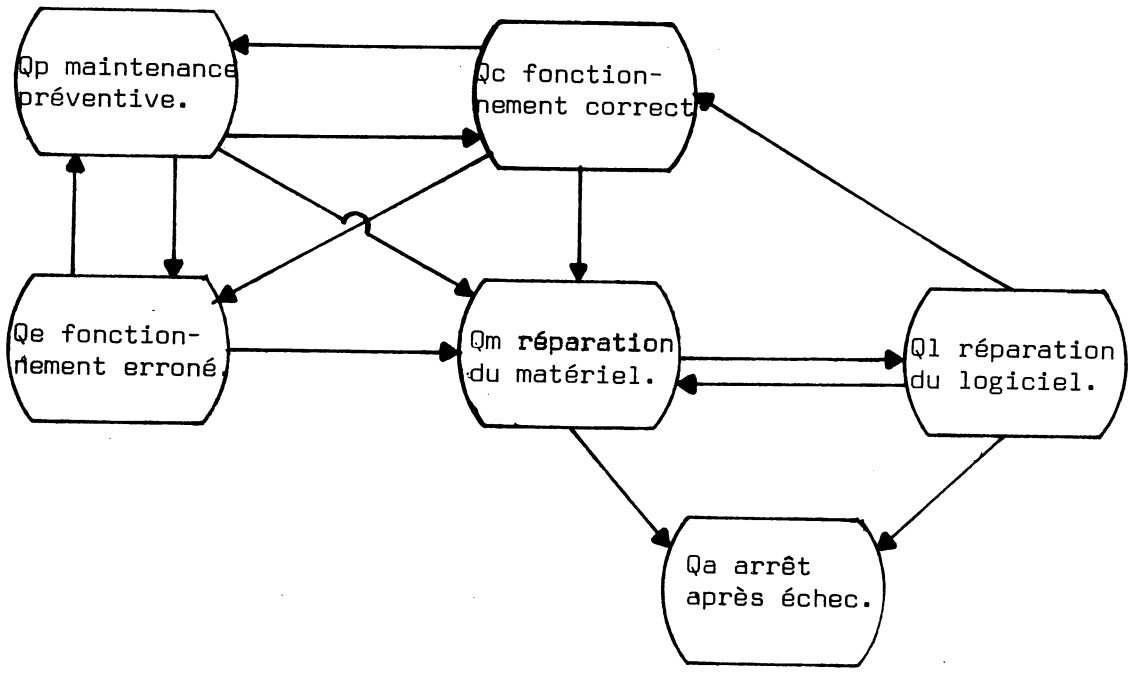


Fig. I - 1 Modèle Général

II - 2 EVOLUTION DU SYSTEME

Le système passe :

- de l'état Qc à Qe dès l'apparition d'une erreur non détectée,
- de l'état Qc à Qm, si une panne est détectée avant l'apparition d'une erreur, ou si une erreur est détectée avant propagation de l'erreur,
- de l'état Qe à Qm, dès qu'une erreur est détectée au cours du fonctionnement erroné,
- de Qc ou Qe à Qp, sur des critères (périodicité de la maintenance préventive...) indépendants du fonctionnement correct ou incorrect,
- de Qp à Qc s'il n'y a pas de panne,
- de Qp à Qm, si la maintenance détecte une panne,
- de Qp à Qe, si une panne n'a pas été détectée par la maintenance, le taux de couverture n'étant jamais égal à 1.

III - COMPOSANTES DE LA SURETE DE FONCTIONNEMENT

Fiabilité : la fiabilité R(T) à l'instant T est la probabilité que le système soit en état de bon fonctionnement, c'est à dire qu'aucune erreur ne soit produite de l'instant 0 à l'instant T. La fiabilité à l'instant T d'un système, est la probabilité que le système soit à l'état Qc ou à l'état Qp, entre 0 et T.

$R(T) = \text{Proba}(\text{état}(t) = Qc \vee \text{état}(t) = Qp, \forall\ t \in [0,T])$.

Disponibilité : la disponibilité A (T) à l'instant T est la probabilité que le système soit en état de bon fonctionnement à l'instant T, c'est à dire qu'il soit à l'état Qc ou Qp à l'instant T:

$A(T) = \text{Proba}(\text{état}(T) = Qc \vee \text{état}(T) = Qp)$.

Sécurité : la sécurité S (T) à l'instant T est la probabilité que le système n'ait pas communiqué de valeurs erronées au monde extérieur, c'est à dire qu'il n'y ait pas eu d'erreurs non détectées de l'instant 0 à l'instant T, ou que dès qu'il y a eu erreur, le système ait été arrêté.

La valeur de la sécurité est la probabilité que le système ne soit pas passé à l'état Qe, entre 0 et T.

$S(T) = \text{Proba}(\text{état}(t) \neq Qe, \forall\ t \in [0,T])$.

Crédibilité : la crédibilité C (T) à l'instant T est la probabilité qu'il n'y ait pas d'erreurs non détectées à l'instant T. Sa valeur est donc la probabilité que le système ne soit pas à l'état Qe à l'instant T.

$C(T) = \text{Proba}(\text{état}(T) \neq Qe)$

La disponibilité et la crédibilité n'ont une valeur différente de la fiabilité et de la sécurité, respectivement, que si le système est réparable (le taux de transition de Q1 à Qc est différent de 0).

Maintenabilité du matériel : ce paramètre concerne la maintenance curative du matériel. La maintenabilité M(T) est la probabilité que le système ne soit plus à l'état Qm à l'instant T, sachant qu'il y était à 0:
$M(T) = 1 - Proba(\ état(t) = Qm\ ,\forall\ t\ \epsilon[0,T]\ )$

Réparabilité du logiciel : ce paramètre concerne la remise en état du logiciel. La réparabilité L(T) est la probabilité que le système ne soit plus à l'état Q1 à l'instant T, sachant qu'il y était à l'instant 0 :
$L(T) = 1 - Proba\ (état(t) = Q1\ ,\forall\ t\ \epsilon[0,T]\ )$

IV - APPLICATION AUX SYSTEMES AVEC DETECTION MATERIELLE

L'étude de la signification des composantes de la sûreté de fonctionnement sera faite pour des systèmes avec détection faite par du matériel. Ces systèmes peuvent avoir une redondance sélective (codes) : le matériel utilisé pour la détection (ou contrôleur) comprend alors les bits supplémentaires nécessités par le codage et le contrôleur de code proprement dit. Dans le cas de redondance massive, le contrôleur comprend le comparateur ainsi que les unités redondantes. Le contrôleur étant matériel, peut donc tomber en panne ; ce contrôleur, étant assez important, surtout dans le cas de redondance massive, on ne peut négliger cette éventualité.

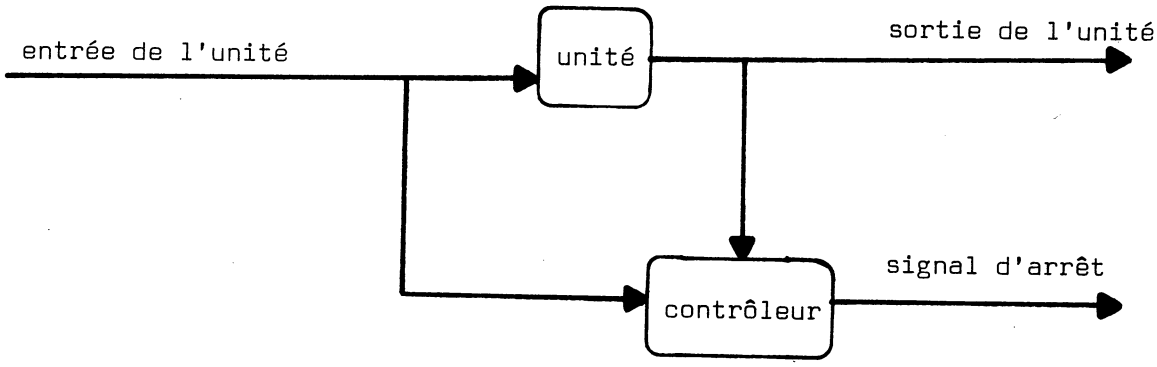


Fig I - 2 Système avec détection matérielle

a) hypothèse 1 :le contrôleur détecte toute erreur .

Un tel système se décompose en trois éléments :

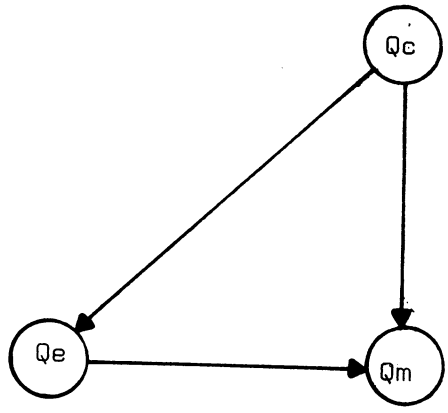
- l'unité (ou matériel minimum nécessaire pour réaliser la fonction demandée), qui peut être soit en panne (état défectueux), soit sans panne (état sain),
- le contrôleur (composé de tout le matériel non nécessaire fonctionnellement) qui prend lui aussi deux états : sain ou défectueux,
- le signal d'arrêt, qui peut prendre deux valeurs : marche et arrêt. (le matériel nécessaire pour la réalisation de ces valeurs, est inclus dans le contrôleur).

Un tel système a donc huit états. Deux de ses états ont une probabilité nulle.

		CONTROLEUR	
SIGNAL	UNITE	SAIN	DEFECTUEUX
MARCHE	SAIN	1	2
	DEFECTUEUX		3
ARRET	SAIN		4
	DEFECTUEUX	5	6

On considérera le système comme non réparable, c'est à dire que la disponibilité est égale à la fiabilité et la crédibilité est égale à la sécurité.

Toute erreur étant détectée, (quand le contrôleur est sain), on étudie une partie du modèle général d'évolution, défini enII - 1, en détaillant les différentes possibilités :



Qc = {1,2}
Qe = {3}
Qm = {4,5,6}

Soit Pi la probabilité de l'état i;
La fiabilité R du système est égale à la probabilité des états 1 et 2.

$$R(T) = P1 + P2$$

La sécurité S est la probabilité que l'unité soit saine ou que le système soit arrêté.

$$S(T) = \sum_{i \neq 3} Pi$$

Deux états mettent particulièrement en évidence l'influence des pannes du contrôleur :

- état 3 : le contrôleur n'a pas rempli son rôle. Il y a possibilité de sorties erronées ; l'état 3 est un état dangereux,
- état 4 : le contrôleur a arrêté le système abusivement ; cet état sera appelé état d'arrêt abusif.

Dans les états 2 et 6, les pannes du contrôleur n'ont pas d'influence sur le système.

Si Ru est la fiabilité de l'unité, Rc la fiabilité du contrôleur, on a :

$P1 = Ru.Rc$

$P2 + P4 = Ru.\overline{Rc}$

$P3 + P6 = \overline{Ru}.\overline{Rc}$

$P5 = \overline{Ru}.Rc$

Il est difficile d'évaluer les états 3 (dangereux) et 4 (arrêt abusif); il faut pour cela connaître exactement les circuits qui composent le contrôleur, et pouvoir déterminer les pannes du contrôleur qui conduisent à la mise du signal à la valeur "arrêt" et celles qui le maintiennent à la valeur "marche".

A l'heure actuelle, la tendance est de concevoir des contrôleurs, tels que toute panne du contrôleur conduise à un signal "arrêt".(self-checking checkers [4] ,[5], [6]).

Cela minimise (ou même annule) P2 et P3. On conçoit donc ainsi des systèmes très sûrs (P3 ≃ 0) mais tels que la probabilité d'arrêt abusif est importante.

b) hypothèse 2 : une erreur est détectée avec la probabilité d.

Le système se décompose aussi en trois éléments, mais l'unité peut prendre trois états :

- état sain,
- état défectueux non détectable,
- état défectueux détectable.

Le système a donc douze états, trois d'entre eux ayant une probabilité nulle.

		CONTROLEUR	
SIGNAL	UNITE	SAIN	DEFECTUEUX
MARCHE	SAIN	1	2
	DEFECTUEUX NON DETECTABLE	7	3_a
	DEFECTUEUX DETECTABLE		3_b
ARRET	SAIN		4
	DEFECTUEUX NON DETECTABLE		6_a
	DEFECTUEUX DETECTABLE	5	6_b

Si on se reporte au modèle général d'évolution, on obtient :

$Qc = \{1,2\}$

$Qe = \{7,3_a,3_b\}$

$Qm = \{4,5,6_a,6_b\}$

On note P_i^1 la probabilité de l'état i_1 à l'instant T.

La fiabilité du système est donc :

$R(T) = P1 + P2$

La sécurité du système vaut :

$S(T) = 1 - (P_3^a + P_3^b + P_7)$

L'état d'arrêt abusif est l'état 4.

Si Ru est la fiabilité de l'unité à l'instant T, Rc celle du contrôleur, on obtient :

$Ru.Rc = P1$

$Ru.\overline{Rc} = P2 + P4$

$\overline{Ru}.d.Rc = P5$

$\overline{Ru}.d.\overline{Rc} = P_3^b + P_6^b$

$\overline{Ru}.\overline{d}.Rc = P_7$

$\overline{Ru}.\overline{d}.\overline{Rc} = P_3^a + P_6^a$

Dans ce cas aussi, l'évaluation de l'arrêt abusif et de la sécurité ne peut pas se faire en connaissant uniquement la fiabilité de l'unité et du contrôleur. Une étude détaillée du contrôleur est nécessaire :

- pour déterminer d,
- pour déterminer la proportion des pannes qui positionnent le signal à la valeur "arrêt".

V - CONCLUSION

Il est donc clair que l'adjonction de matériel de détection dans un système doit être évaluée de façon précise par l'analyse du matériel supplémentaire. On recherchera un compromis entre la sécurité (probabilité de ne pas être dans un état dangereux) et la durée de fonctionnement du système (probabilité de ne pas être dans un état d'arrêt abusif).

CHAPITRE II

ETUDE DU PARTITIONNEMENT ET DE LA DEGRADATION PROGRESSIVE DE PUISSANCE.

I - INTRODUCTION

Après description des stratégies de partitionnement d'un système, divers types de redondance permettant la dégradation de puissance après panne sont définis et modélisés. Les fonctions permettant l'évaluation de la puissance d'un système à dégradation sont définies ; cette évaluation est faite

- pour des systèmes redondants modulaires. Cela permet de sélectionner un schéma de redondance interessant.
- pour un système modulaire non redondant, en vue de déterminer le nombre de modules nécessaires pour une puissance minimale fixée.
- pour un système "pipe-line".

Diverses tentatives d'évaluation de la dégradation de puissance ont été faites. En (7), l'auteur est interessé à la probabilité de terminaison d'une tâche. Cette grandeur nécessite l'évaluation des performances du système dans chacune des configurations possibles. En (8), l'hypothèse faite est que le système entier est arrêté après chaque panne, et n'est remis en route qu'après reconfiguration du matériel et du logiciel. Cette hypothèse n'est valable, que pour des systèmes non redondants, du type centre de calcul. Dans un système redondant, la reconfiguration ne nécessite pas l'arrêt total du système, la propagation des erreurs étant très limitée. Dans les systèmes tolérant des erreurs momentanées (type centrale téléphonique) l'arrêt du système n'est pas nécessaire. Nous supposerons donc, dans la suite de ce chapitre, que le système n'est pas arrêté, après détection d'une panne. En (9), l'auteur s'est intéressé à l'évaluation de l'indisponibilité des systèmes "polymorphes", pour justifier le partitionnement d'un système en sous-systèmes, puis des sous-systèmes en modules.

II - PARTITIONNEMENT D'UN SYSTEME

II - 1 DEFINITION

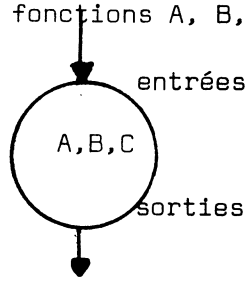
Un système peut être partitionné en unités coopérant entre elles de différentes façons :

- partitionnement en unités équivalentes ou modules. Chaque module peut effectuer toutes les fonctions du système. La puissance désirée est obtenue par la coopération des modules ; le système fonctionne en partage

de trafic. Une tâche est confiée à un module. Le système est dit modulaire.

- partitionnement en unités différentes ou sous-systèmes. Chaque sous-système réalise une fonction. Le système est dit "distribué". L'exécution d'une tâche nécessite l'intervention de plusieurs ou de tous les systèmes. Un sous-système peut être modulaire.

Soit un système réalisant les fonctions A, B, C :



II - 1 a PARTITIONNEMENT EN MODULES

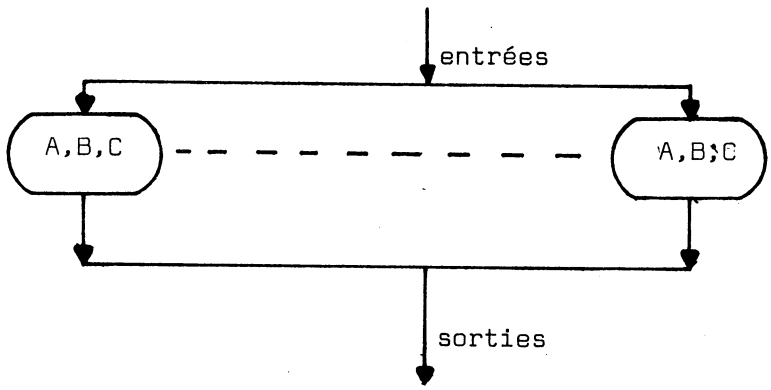


Fig. II - 1

II - 1 b PARTITIONNEMENT EN SOUS-SYSTEMES

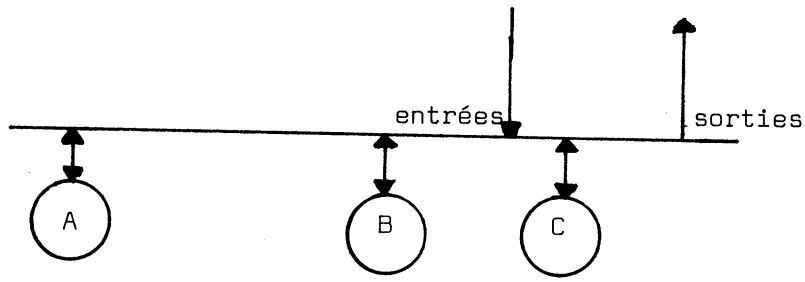


Fig. II - 2

- cas particulier 1 : architecture "pipe-line" : les fonctions sont exécutées dans l'ordre A, B, C.

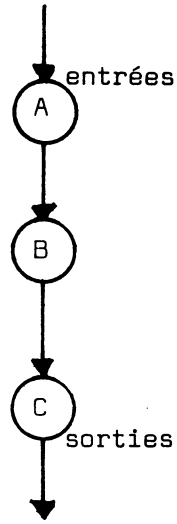


Fig. II - 3

- cas particulier 2 : systèmes hiérarchisés (gestion de banques, téléphone)

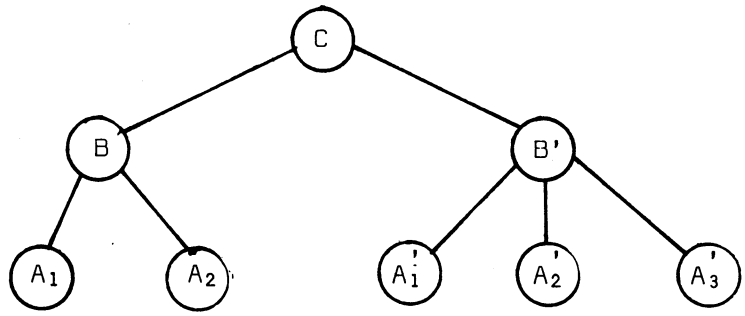


Fig. II - 4

Par exemple, dans un système de gestion de banque, les unités A sont les terminaux (intelligents) d'agences, les unités B sont les centres au niveau d'une ville, l'unité C le centre régional ou national.

Dans un système téléphonique, les unités A sont les unités de raccordement, B les marqueurs, C le réseau de connexion, etc...

Les unités B et B', par exemple, ne sont pas des modules : elles font les mêmes tâches, mais ne disposent pas des mêmes données (elles sont raccordées à des agences ou des abonnés différents).

II - 1 c PARTITIONNEMENT MIXTE

On peut décomposer les sous-systèmes en modules :

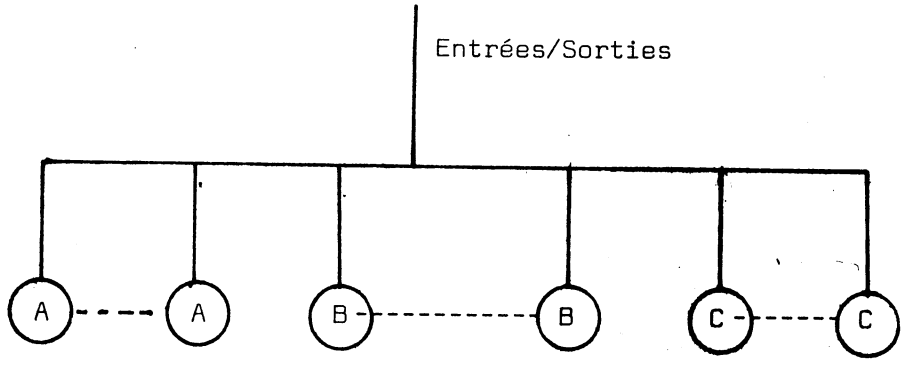


Fig. II - 5

Exemple : systèmes multimicroprocesseurs

Les systèmes multimicroprocesseurs peuvent comporter un certain nombre de sous-systèmes spécialisés (sous-systèmes de traitement, sous-systèmes entrées/sorties...) qui sont composés de plusieurs modules.

Remarque : Impact sur la puissance et le coût en l'absence de pannes

Evaluer l'impact du partitionnement sur la puissance et le coût en l'absence de pannes ne fait pas partie de notre propos.

Mentionnons seulement que chaque sous-système ou module peut être moins puissant et doit être moins coûteux que le système à réaliser. Si un système est décomposé en deux modules, et qu'il doit avoir une puissance P et un coût C; chaque module doit :

- avoir un coût inférieur à C/2, puisqu'il y aura du matériel d'interconnexion supplémentaire,
- avoir une puissance réelle supérieure à P/2, puisque le partage du trafic et d'information conduira à une puissance apparente dégradée.

II - 2 IMPACT DU PARTITIONNEMENT SUR LA SECURITE ET LA TOLERANCE AUX PANNES

II - 2 a PARTITIONNEMENT ET IMPLEMENTATION DE REDONDANCE

L'implémentation de redondance est facilitée par un partitionnement du système : pour des raisons de synchronisation et de minimisation des effets des erreurs , une redondance au niveau d'un système complet est difficile à maîtriser.

L'implémentation de redondance à un niveau fin (registres, UAL, etc) est peu compatible avec l'intégration croissante des composants. Elle nécessite l'étude de boitiers spécialisés pour la sécurité, et par conséquent coûte très cher. Le coût d'un boitier tripliqué est bien supérieur au triple du coût.

de la réalisation simplex : il faut y ajouter les frais de mise en oeuvre.

La tendance actuelle est donc d'implémenter la redondance à un niveau moyen (microprocesseurs, bancs-mémoire...).

Il est donc assez simple de rendre sûr un système distribué : la redondance est implémentée au niveau sous-système ou module.

Les différentes politiques de redondance au niveau interprocesseurs sont :

- duplication figée,
- duplication dynamique,
- triplication dynamique.

La duplication figée employée dans (10), (11), (12), consiste à jumeler par construction deux processeurs. Quand il y a désaccord, on supprime les deux processeurs jumelés sans chercher lequel est en panne. Cette solution est simple à réaliser.

La duplication dynamique (4) consiste à former les couples de processeurs par le logiciel. Quand il y a désaccord, on fait appel à un troisième processeur pour déterminer le processeur en panne. Celui-ci est supprimé ; l'autre processeur est jumelé à un processeur libre s'il y en a, sinon il est déclaré libre. S'il y a n couples, après une panne, il n'y a plus que n-1 couples ; après deux pannes il y en a toujours n-1 (au lieu de n-2 pour la duplication figée).

La triplication dynamique suit le même principe : s'il y a désaccord, le processeur en panne est déterminé par vote majoritaire puis supprimé. S'il existe un processeur libre, on peut reformer une triade. Sinon, les deux processeurs restant sont déclarés libres et mis en réserve. Après trois pannes, on dispose de n-1 triades sur les n dont on disposait initialement. On ne permet pas qu'un processus ne soit exécuté plus d'un certain temps, très court, que par deux processeurs ; en cas de seconde panne, on ne pourrait plus rien faire (pas de points de reprise prévus). C'est donc très différent du TMR, solution employée au niveau fin.

II - 2 b PARTITIONNEMENT ET DEGRADATION PROGRESSIVE

Un système admet une dégradation progressive en présence de pannes lorsqu'il peut être automatiquement reconfiguré, logiciellement et matériellement après la détection d'une panne du matériel, sans avoir recours à du matériel en réserve.

Dans un système distribué, la reconfiguration matérielle consiste à supprimer l'unité défectueuse ; la reconfiguration logicielle consiste à adapter le logiciel à la nouvelle structure matérielle et à assurer la répartition des tâches sur les unités restantes. Le système peut donc continuer à fonctionner (à assurer toutes ses fonctions) quoique avec des performances réduites. Ceci

implique que la dégradation progressive n'est utilisable que pour des systèmes permettant cette perte de performance (centre téléphonique, centre de calcul).

Dans ce cas de systèmes répartis, la reconfiguration logicielle est généralement complexe ; en effet, chaque sous-système réalisant une (sous) fonction spécifique, dispose d'informations particulières et/ou d'une structure matérielle spécialisée. Il peut être difficile -voire impossible- de faire exécuter les tâches d'un sous-système par un autre sous-système. Par contre, dans le cas de systèmes modulaires, les modules sont équivalents et chaque module est capable d'assurer l'ensemble des fonctions du système. Il s'ensuit que la reconfiguration logicielle est simple : la charge du système est équitablement répartie sur les modules restants. On s'intéressera donc par la suite aux systèmes modulaires.

II - 2 c LA RECONFIGURATION

Soit un système modulaire travaillant en partage de trafic, représenté comme suit :

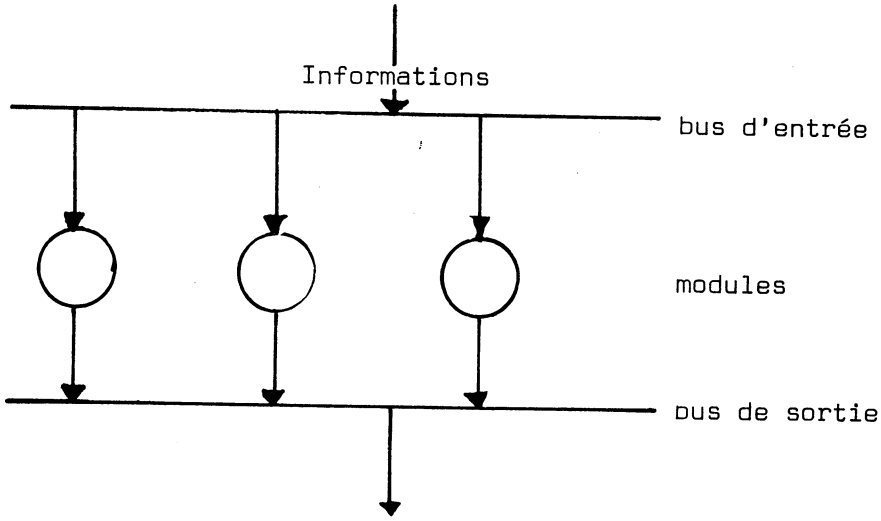


Fig II - 6

Les mécanismes d'entrée et de sortie de l'information sur les modules peuvent être réalisés par un système complexe d'interconnexions, incluant en particulier le mécanisme d'allocation des tâches. Mais, pour que le partage de trafic soit possible, ils doivent être équivalents à des bus du type bus d'entrée et bus de sortie (Cf. Fig. II-6), où chaque module peut accéder à toutes les informations.

Dans une structure de ce type, la reconfiguration matérielle consiste à bloquer les interfaces du module défectueux avec le reste du système (isolation);

la reconfiguration logicielle consiste à modifier l'algorithme d'allocation des tâches : ceci présuppose une flexibilité de cet algorithme.

Après reconfiguration, les modules restants (et opérationnels) réalisent la totalité des tâches.

III - MODELISATION DE LA DEGRADATION PROGRESSIVE DANS UN SYSTEME MODULAIRE REDONDANT

III - 1 MODELE MATERIEL

La modélisation de la dégradation progressive sera faite pour une structure très simple, la plus générale.

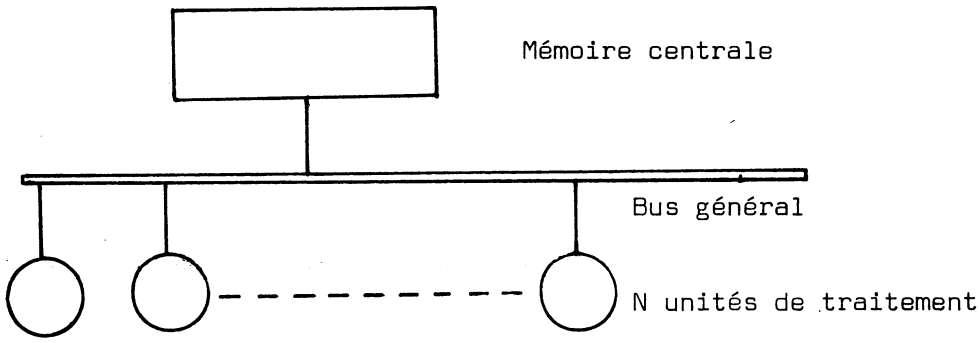


Fig. II - 7

Les unités de traitement peuvent être depuis des microprocesseurs jusqu'à des calculateurs complets avec mémoire locale et contrôleurs d'E/S.

On ne considère que les pannes des unités de traitement. La mémoire centrale et le bus général sont considérés comme beaucoup plus fiables, ce qui est une hypothèse raisonnable si le système est conçu de manière cohérente. En se reportant au modèle général du chapitre I (Fig. I - 1), ces systèmes peuvent prendre 3 types d'états :

- états type Q_c : fonctionnement correct,
- états type Q_m : une erreur a été détectée : la panne est localisée, le module en panne est déconnecté,
- états type Q_1 : le logiciel est réparé et modifié pour s'adapter à la nouvelle configuration.

On prend donc comme hypothèses :

- toute erreur est détectée instantanément ou quasi instantanément (détection en fin de séquence). En tous cas, il ne peut y avoir d'erreur non détectée dangereuse pour le logiciel ou l'application. Nous nous intéres-

serons donc uniquement aux systèmes redondants.

- il n'y a pas de maintenance préventive,
- il n'existe pas de pannes ou d'erreurs non localisables ou non réparables.

III - 2 DUPLICATION FIGEE (PUISSANCE MAXIMALE)

Il existe N+1 états disponibles de type Q_c. Le système est à l'état Q_c^i (i^e état de type Q_c) quand i couples de 2 processeurs sont en état de marche. On obtient le schème :

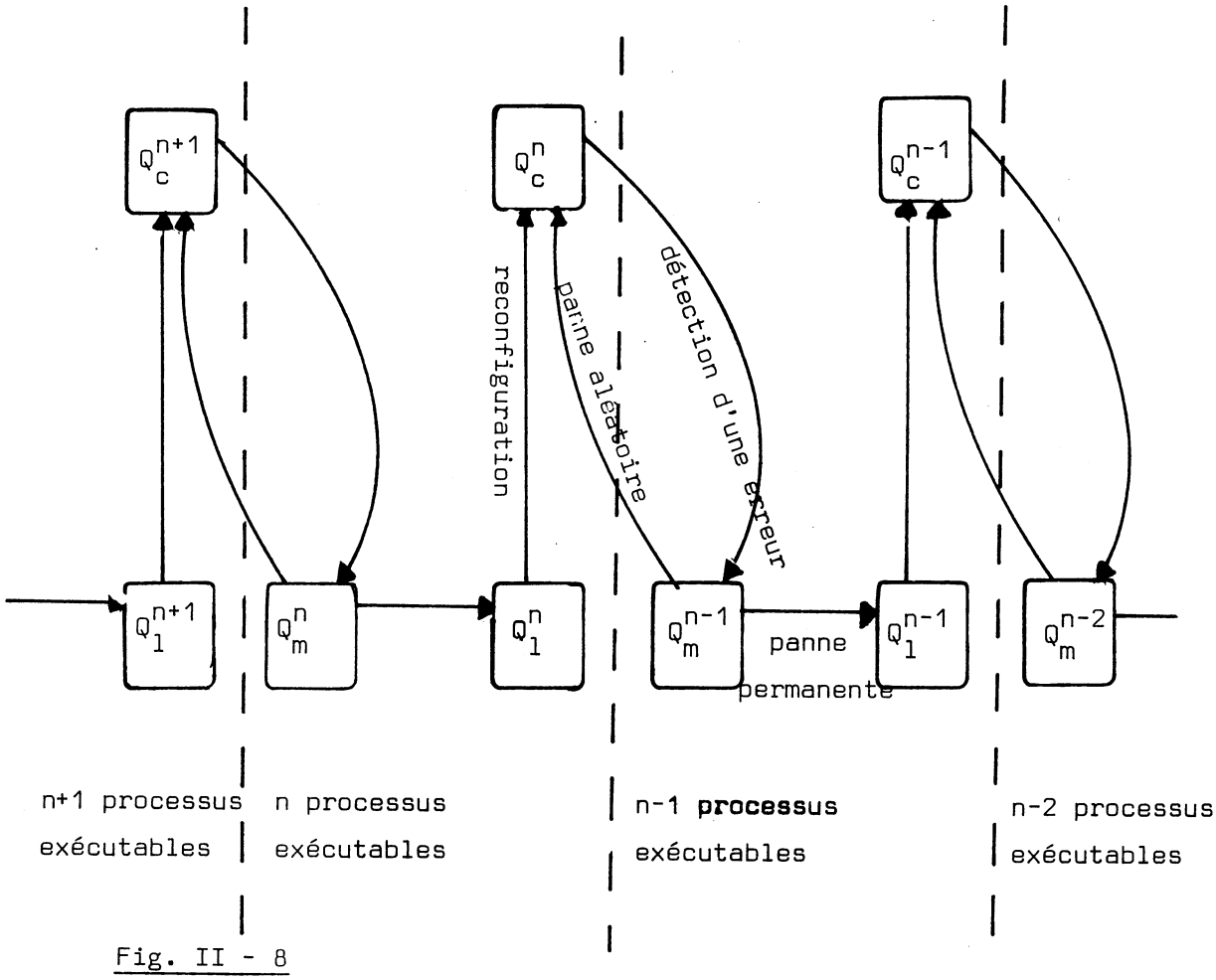


Fig. II - 8

Un ensemble d'états Q_c^n, Q_m^n, Q_1^n représente l'état où la puissance du système est n. Un état Q_c^n est un état de traitement normal: tous les couples de processeurs exécutent normalement un processus. On passe de l'état Q_c^n à l'état Q_m^{n-1} dès la détection d'une erreur; dans l'état Q_m^{n-1} la déconnexion du couple en panne est effectuée; les ressources réservées par ce couple sont libérées. Eventuellement, le logiciel est reconfiguré (suppression du couple sur la table des ressources etc...). Selon les cas, le processus exécuté par ce couple sera attribué à un autre couple oisif, si il y en a un,

ou seulement relancé par le watch-dog. Une fois la reconfiguration faite, le système passe à l'état Q_c^{n-1} .

III - 3 DUPLICATION DYNAMIQUE

Le modèle est plus complexe, traduisant la complexité plus importante de la reconfiguration.

Dans chaque état où n processus sont exécutables, deux cas se présentent :

- exactement 2n processeurs sont en état de marche,
- 2n+1 processeurs sont en état de marche, 2n étant en service, le dernier étant en réserve.

Il y a deux types de reconfigurations, donc deux types d'états Q_m. Dans tout état Q_m, le processeur en panne est déterminé.

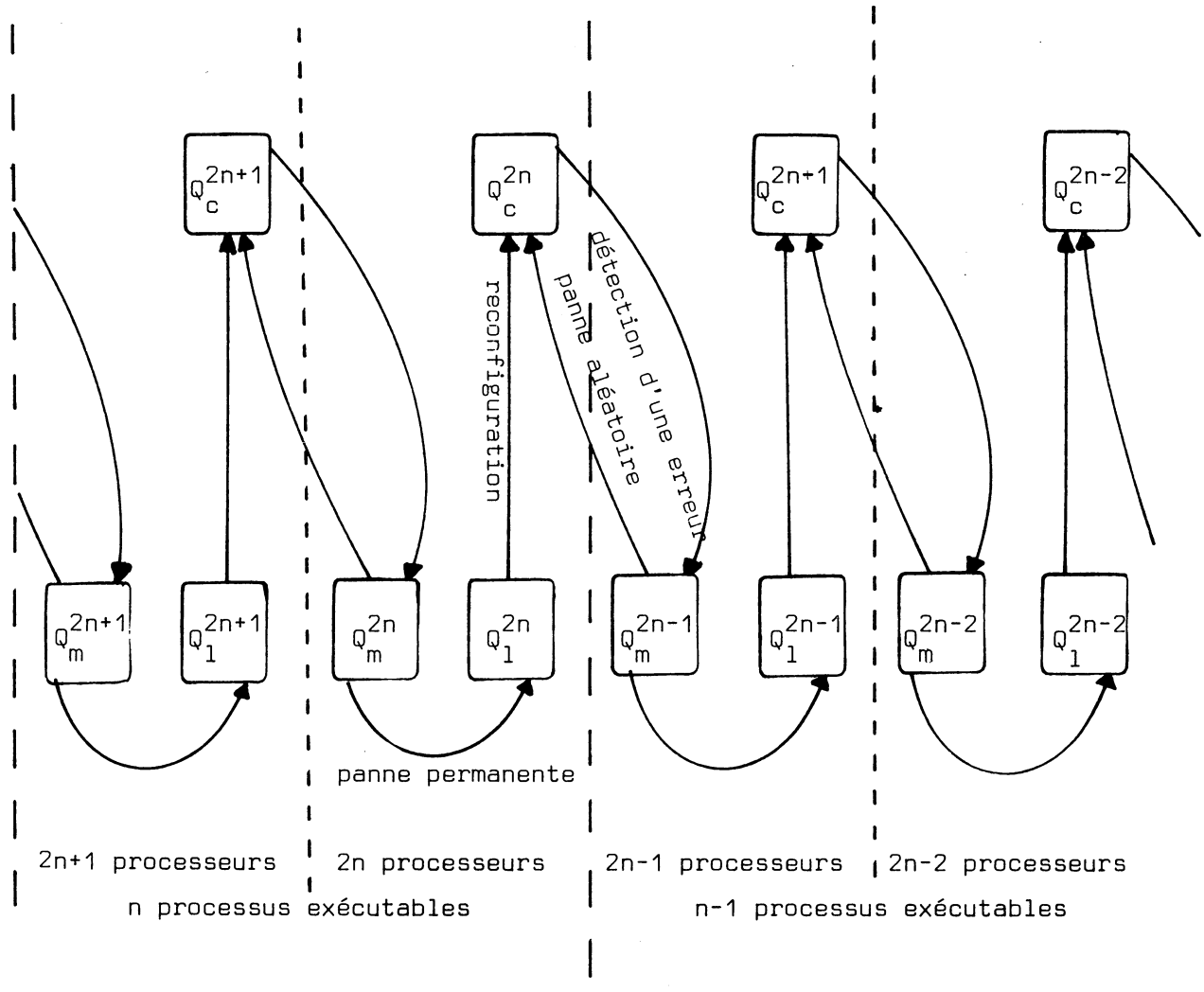


Fig. II - 9

L'ensemble des états $\{Q_c^{2n+1}, Q_c^{2n}, Q_m^{2n+1}, Q_m^{2n}, Q_1^{2n+1}, Q_1^{2n}\}$ représente l'état où la puissance du système est n.

Un état Q_c^{2n+1} ou Q_c^{2n} est un état où les n couples exécutent normalement un processus.

On passe d'un état Q_c^{2n+1} (Q_c^{2n}) à un état Q_m^{2n} (Q_m^{2n-1}) par la détection d'une erreur. Dans tout état Q_m on détermine si la panne est aléatoire ou non par réessais. Dans les états Q_m^{2n} et Q_m^{2n+1} la puissance du système est n.

Si la panne est transitoire le système passe de l'état Q_m^{2n} ou Q_m^{2n-1} à l'état Q_c^{2n+1} ou Q_c^{2n}.

Si la panne est permanente, le système passe de l'état Q_m^{2n} ou Q_m^{2n-1} à l'état Q_1^{2n} ou Q_1^{2n-1}.

Il y a donc deux types différents d'état Q_m :

état Q_m^{2n} : le système dispose de

- n-1 couples de processeurs en état de marche,
- 1 couple de processeurs dont 1 est en panne,
- 1 processeur en état de marche en réserve.

Il fait appel au processeur en réserve pour déterminer quel processeur du couple est en panne, puis après déconnexion de ce processeur, un couple est formé avec le processeur restant et le processeur en réserve.

état Q_m^{2n-1} : le système dispose de

- n-1 couples de processeurs en état de marche,
- 1 couple de processeurs dont 1 est en panne.

Deux solutions sont possibles :

- interruption d'un processeur actif qui détermine quel processeur est en panne. Après deconnexion du processeur fautif, le processeur restant est mis en réserve et le processus qu'il exécutait (ou un autre) est mis en attente.
- mise en attente du processus de localisation qui sera traité dès qu'un processeur sera aisif. Après quoi, la procédure sera semblable.

Après reconfiguration du matériel et du logiciel, le système passe dans un état Q_c successeur.

III - 4 TRIPLICATION DYNAMIQUE

Le modèle suit le même principe que celui de la duplication dynamique.

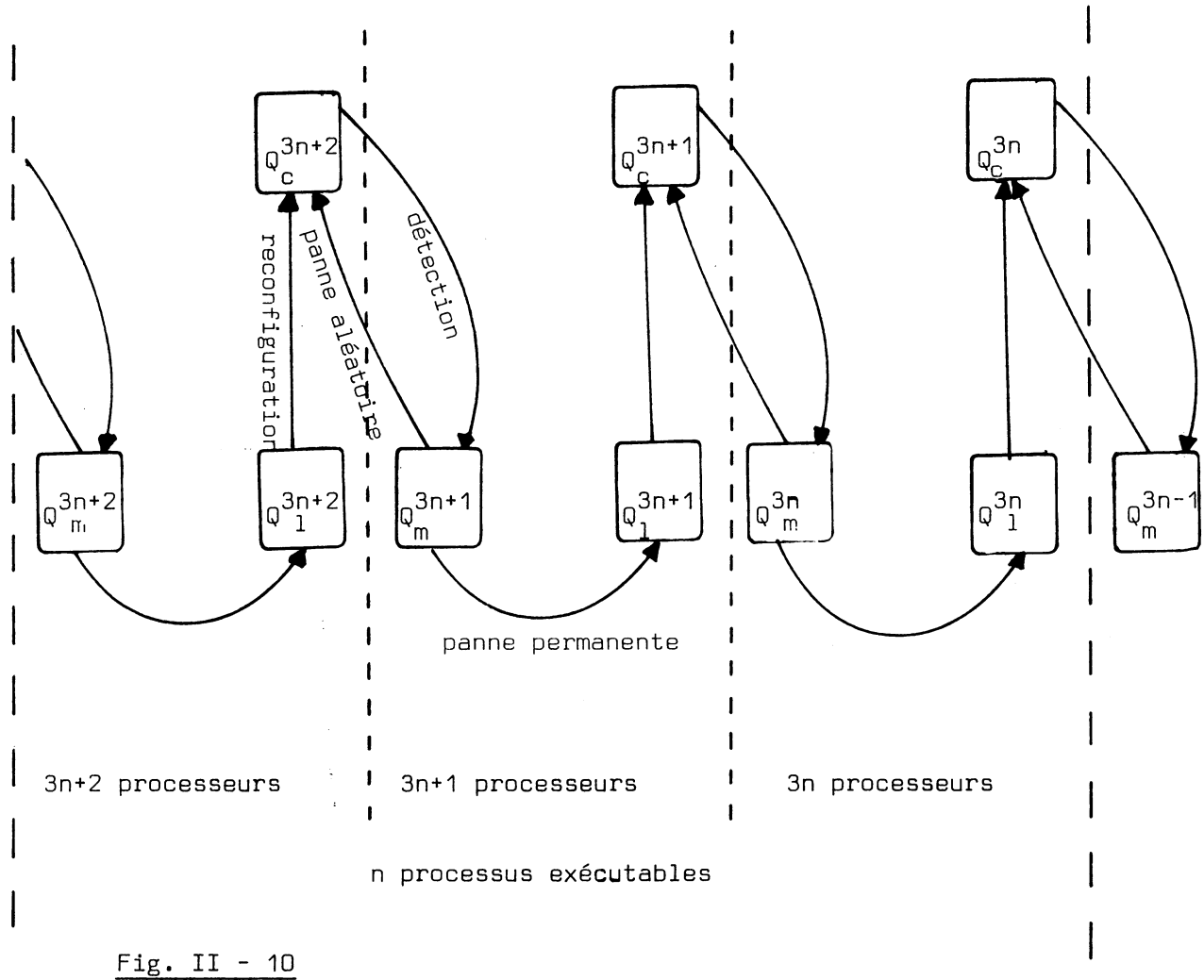


Fig. II - 10

La puissance du système est n dans les états :

Q_c^{3n+2}, Q_c^{3n+1}, Q_c^{3n}

Q_m^{3n+2}, Q_m^{3n+1}, Q_m^{3n}

Q_1^{3n+2}, Q_1^{3n+1}, Q_1^{3n}

Les états Q_c sont des états de traitement normal.

Dans les états Q_m on détermine la nature de la panne par des réessais. Par le vote majoritaire, le processeur défaillant est déterminé.

Il y a deux types d'état Q_m.

Dans les états Q_m^{3n+1} ou Q_m^{3n}, le processeur défaillant est déconnecté et une triade est reformé à l'aide d'un processeur en réserve (il y en a 1 ou 2).

Dans l'état Q_m^{3n-1}, le processus est mis en attente et les deux processeurs en état de marche sont mis en réserve.

IV - EVALUATION DE LA DEGRADATION DE PUISSANCE EN PRESENCE DE PANNES

IV - 1 DEFINITIONS

La puissance normalisée $P(T)$ à l'instant T est le rapport du nombre de processus exécutables en parallèle par le système à l'instant T sur le nombre de processus exécutables à l'instant 0.

Exemple : système modulaire : si ce système est constitué de N modules équivalents, travaillant en partage de trafic, ce système peut exécuter N processus avant panne. Après pannes, s'il reste n modules en état de marche à l'instant T ce système peut exécuter n processus en parallèle. On a donc :

$$P(T) = n / N$$

La μ-fiabilité $R_T(\mu)$ est la probabilité que la puissance normalisée du système soit au moins μ jusqu'à l'instant T :

$$R_T(\mu) = \text{Proba}(P(t) \geq \mu, \forall\ t \in [0,T])$$

La μ disponibilité $A_T(\mu)$ est la probabilité que la puissance normalisée du système soit au moins μ à l'instant T.

$$A_T(\mu) = \text{Proba}(P(T) \geq \mu)$$

Pour un système modulaire on définit de plus :

La puissance $P(T)$ comme le nombre de processus exécutables en parallèle,

La n fiabilité $R_{N,T}(n)$ comme la probabilité que la puissance soit au moins n jusqu'à l'instant T, pour une puissance N à l'instant 0.

$$R_{N,T}(n) = \text{Proba}(P(t) \geq n, \forall\ t \in [0,T]).$$

La n disponibilité $A_{N,T}(n)$ comme la probabilité que la puissance soit au moins n à l'instant T, pour une puissance N à l'instant 0 :

$$A_{N,T}(n) = \text{Proba}(P(T) \geq n).$$

La puissance minimale moyenne du système PMS (T) entre 0 et T :

$$PMS(T) = \sum_{i=0}^{N} \frac{i}{N} \left(R_{N,T}(i) - R_{N,T}(i+1)\right)$$

$R_{N,T}(i) - R_{N,T}(i+1)$ est la probabilité que la puissance minimale entre 0 et T

soit exactement i.

La puissance disponible moyenne du système PDS (T) :

$$PDS(T) = \sum_{i=o}^{N} \frac{i}{N} (A_{N,T}(i) - A_{N,T}(i+1))$$

Des systèmes n'admettant pas de dégradation progressive ont une puissance normalisée égale à 0 ou à 1 et nos définitions concordent avec les définitions classiques de la fiabilité et de la disponibilité.

IV - 2 DEGRADATION DANS UN SYSTEME MODULAIRE

Nous évaluerons la dégradation de puissance pour la structure suivante :

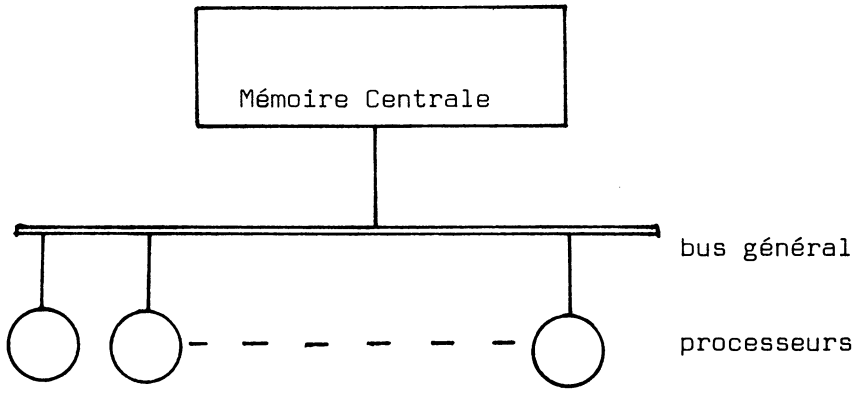


Fig. II - 11

Les cas considérés seront :

- solution non redondante ou simplex,
- duplication dynamique,
- duplication figée,
- triplication dynamique.

Soit R la fiabilité d'un processeur; n processus sont exécutables s'il y a assez de processeurs pour les exécuter.

Nous comparons les différentes solutions à puissance maximale égale, c'est-à-dire qu'à l'instant 0 (où tous les processeurs sont en état de marche) le système peut exécuter N processus en parallèle. Nous avons le tableau suivant :

	n processus au temps T	N processus au temps 0
Non redondant	n processeurs	N processeurs
Duplication figée	n paires de 2 processeurs	N paires de 2 processeurs
Duplication dynamique	2n processeurs	2N processeurs
Triplication dynamique	3n processeurs	3N processeurs

IV - 2 a EVALUATION DE LA n FIABILITE

Pour une solution non redondante :

$$R_{N,T}(n) = \sum_{i=n}^{N} C_N^i R^i (1-R)^{N-i}$$

$$PMS = \frac{1}{N} \sum_{i=o}^{N} i\, C_{2N}^i R^i (1-R)^{N-i} = R$$

Pour le cas de la duplication dynamique :

$$R_{N,T}(n) = \sum_{i=2n}^{2N} C_{2N}^i R^i (1-R)^{2N-i}$$

$$PMS = \frac{1}{N} \sum_{i=o}^{N} i\left[C_{2N}^{2i} R^{2i} (1-R)^{2N-2i} + C_{2N}^{2i+1} R^{2i+1} (1-R)^{2N-2i-1}\right]$$

Pour le cas de la triplication dynamique :

$$R_{N,T}(n) = \sum_{i=3n}^{3N} C_{3N}^i R^i (1-R)^{3N-i}$$

$$PMS = \frac{1}{N} \sum_{i=0}^{N} i\left[\sum_{j=o}^{2} C_{3N}^{3i+j} (1-R)^{3N-(3i+j)} R^{3i+j}\right]$$

Pour le cas de la duplication figée :

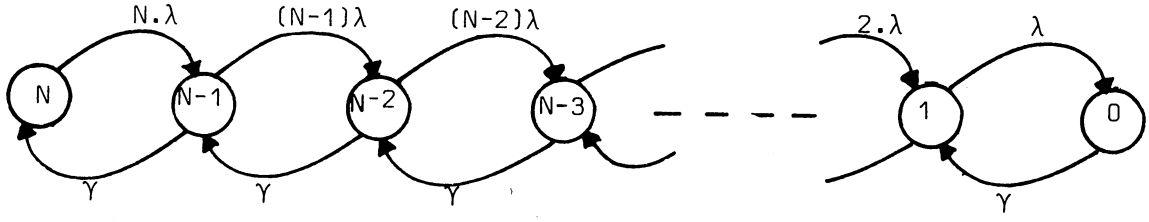
$$R_{N,T}(n) = \sum_{i=n}^{N} C_N^i \, R^{2i} \, (1-R^2)^{N-i}$$

$$PMS = \frac{1}{N} \sum_{i=o}^{N} i \, C_N^i \, R^{2i} \, (1-R^2)^{N-i} = R^2$$

IV ŕ 2 b EVALUATION DE LA n DISPONIBILITE

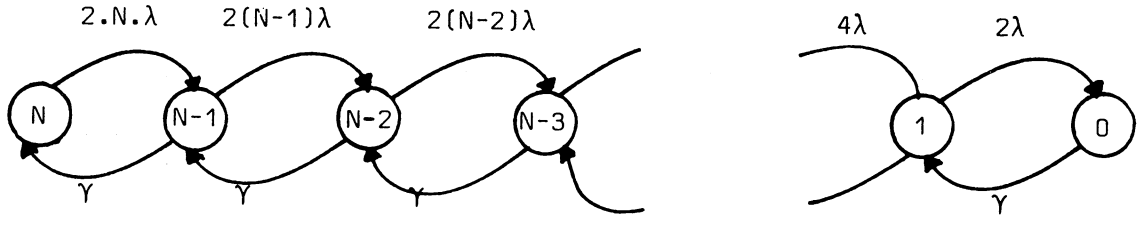
L'évolution du système est décrite par une chaîne de MARKOV homogène à paramètre continu. Les paramètres sont λ, taux de pannes d'un processeur, et γ taux de réparation d'un processeur.

Pour un système non redondant, il y a N+1 états :



Le système a une puissance n à l'état n.

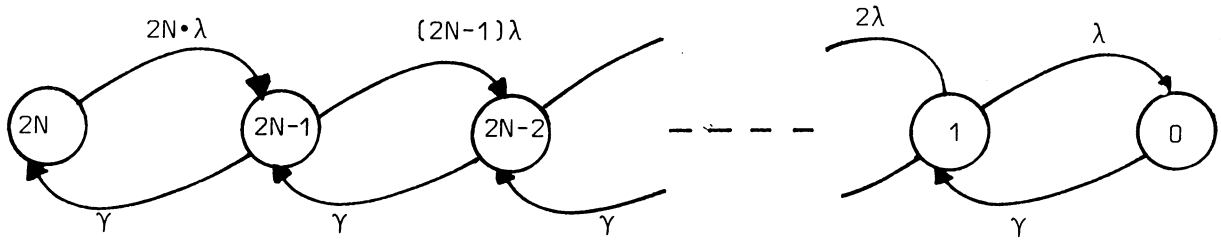
Pour une redondance du type duplication figée, il y a aussi N+1 états :



Le système a une puissance n à l'état n.

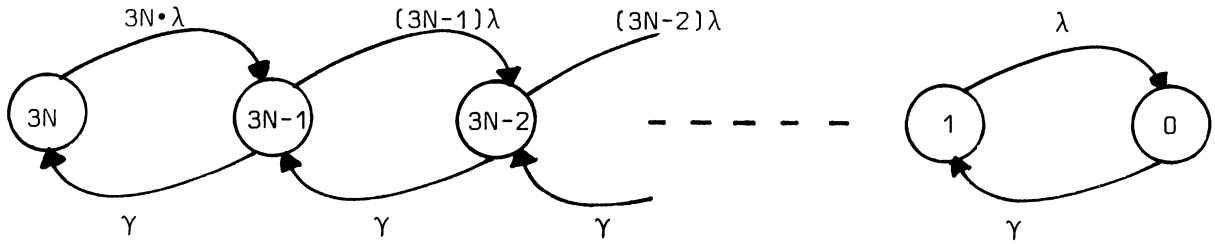
On suppose que le taux de panne d'un processeur déconnecté est nul.

Pour une redondance du type duplication dynamique, il y a 2N+1 états :



Le système a une puissance n aux états 2n et 2n+1.

Pour la triplication dynamique, le système a 3N+1 états :



Le système a une puissance n aux états 3n, 3n+1, et 3n+2.

IV - 2 b COMPARAISON DES DIFFERENTES SOLUTIONS A PUISSANCE EGALE

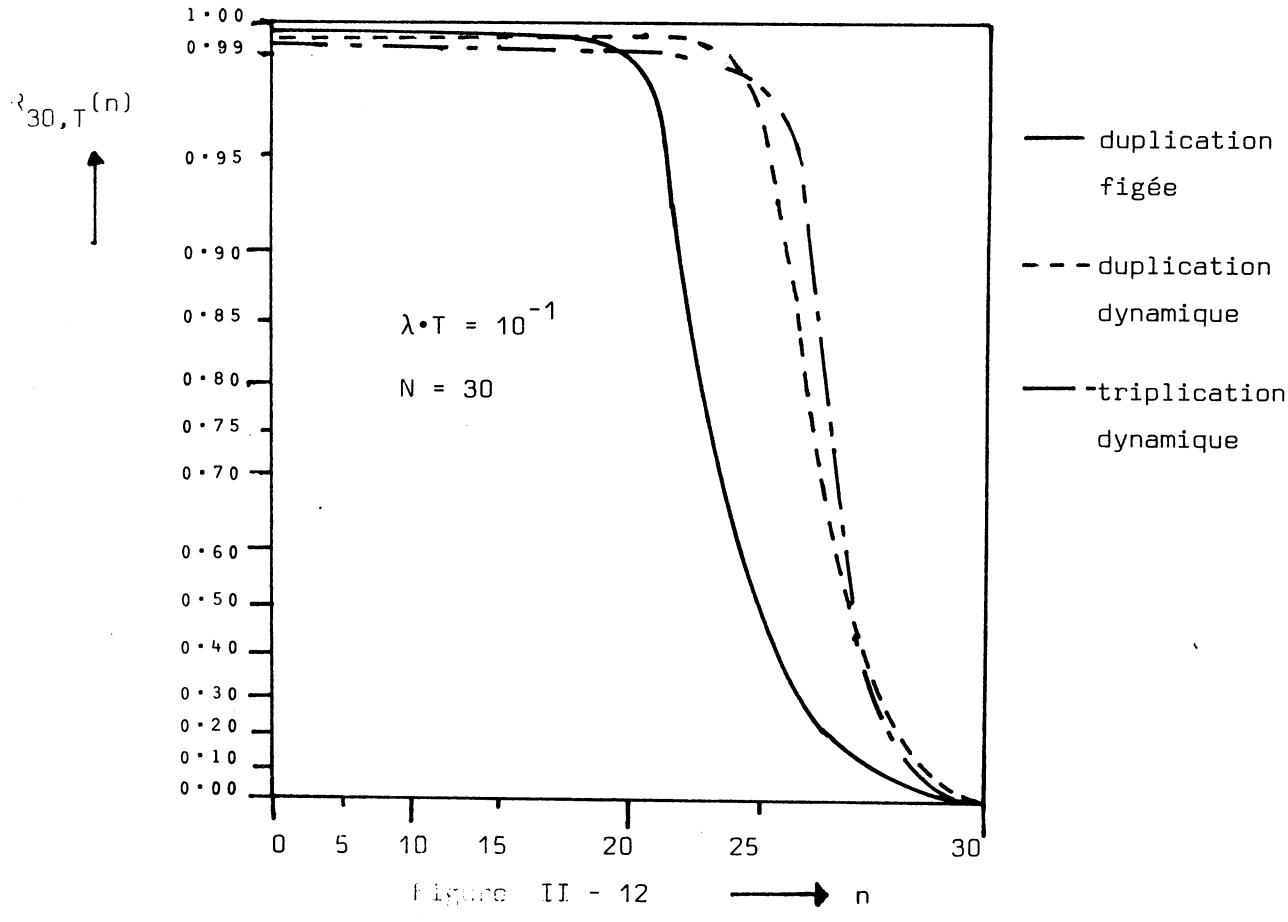


Figure II - 12

La figure II - 12 montre les courbes de la n fiabilité des quatre solutions en fonction de n. N et λT sont fixés (N=39, $\lambda T=10^{-1}$). La n fiabilité est quasi égale à 1 tant que n est plus petit que $\frac{3}{4}$ N. On disposera toujours des trois quarts du système. Ensuite, elle chute rapidement, atteignant 0 dans ce cas (λT est très grand).

La duplication figée est inférieure à la duplication dynamique ; ceci est vrai quel que soit N et λT : la différence décroît si λT décroît, ou si N décroît.

La figure II - 13 montre le graphe du PMS (pourcentage en état de marche du système) en fonction de N. λT est fixé égal à 10^{-2}. Les PMS de la solution non redondante et de la duplication figée sont constants (égal à R dans le premier cas, à R^2 dans le second cas). Les PMS de la duplication et de la triplication dynamiques augmentent avec la puissance du système N.

La duplication dynamique est toujours plus intéressante que la duplication figée et que la triplication. La triplication devient meilleure que la duplication figée si N est grand ou λT grand. Si T est petit (10^{-5}) la triplication n'est jamais meilleure que la duplication figée. Si λT est grand (10^{-1}) la triplication dynamique est meilleure que la duplication figée aussitôt que N est égal à 5.

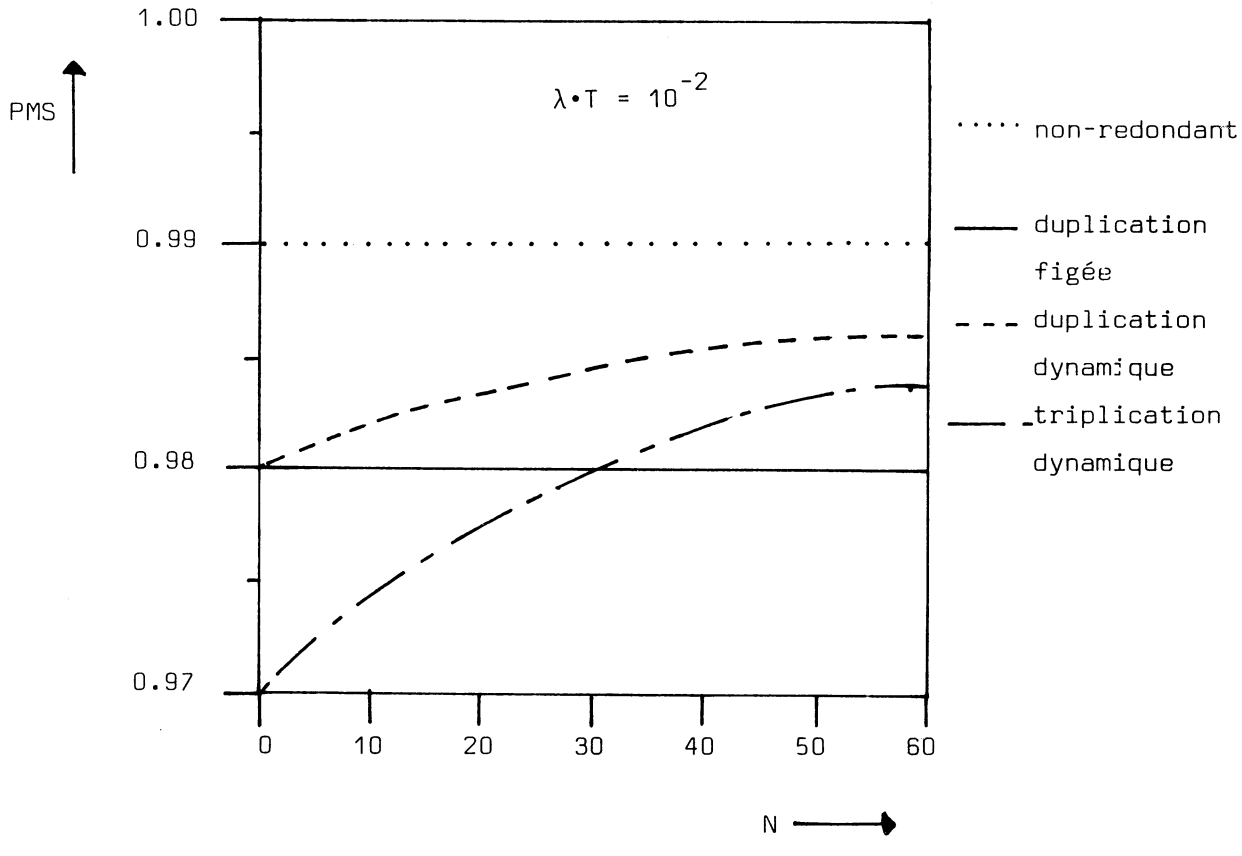


Figure II - 13

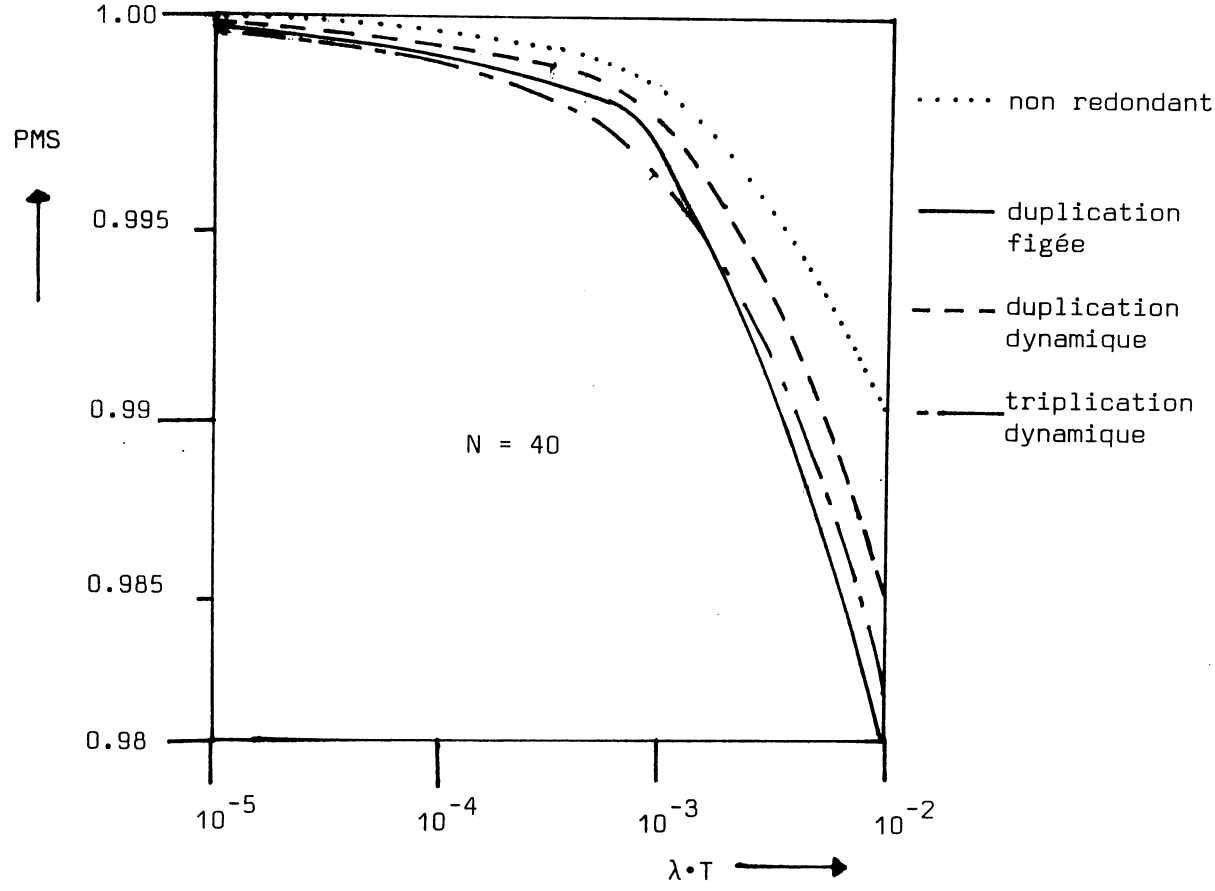


Figure II - 14 .

La figure II - 14 montre le graphe de PMS en fonction de λT. N est fixé à 40. Cette figure conduit aux mêmes conclusions que la figure II - 13.

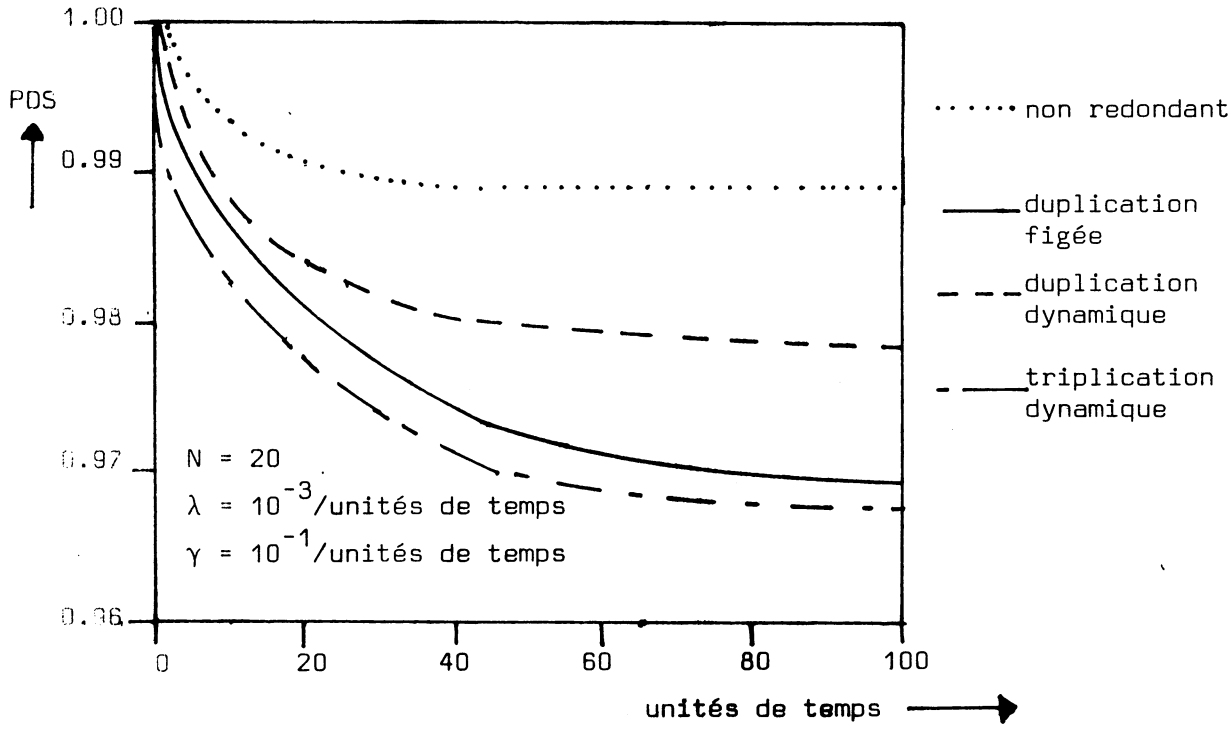


Figure II - 15

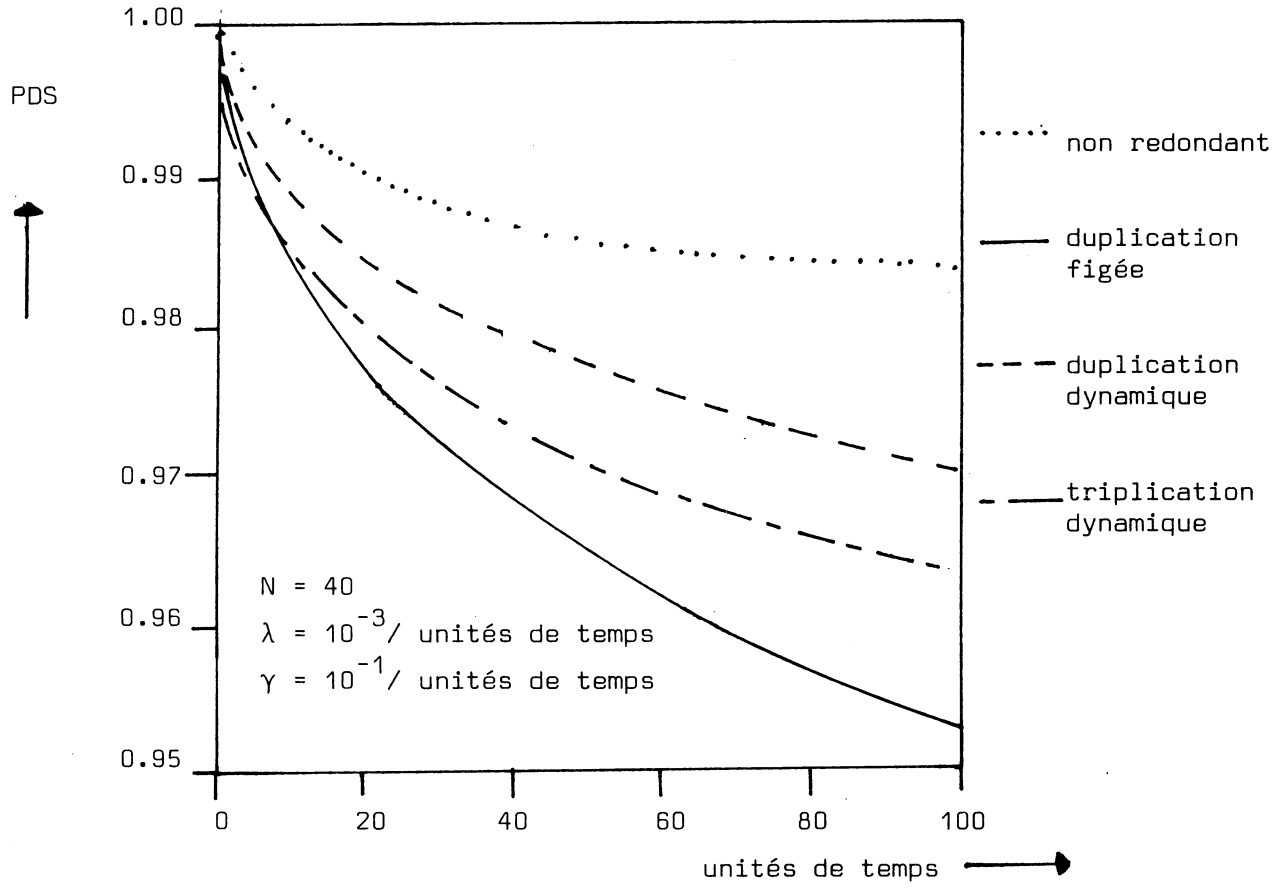


Figure II - 16

Les figures II 15 et II 16 montrent le graphe du pourcentage disponible (PDS) du système en fonction du temps T. Dans la figure 15 la duplication figée est meilleure que la triplication dynamique. Si N ou λ augmente, ou si γ diminue, la triplication devient meilleure que la duplication figée après un certain temps T (Fig. II - 16). La différence entre la duplication dynamique et la duplication figée augmente si N ou λ augmente ou γ diminue.

IV 2 d CONCLUSION

La duplication dynamique est toujours meilleure que la duplication figée. Mais la duplication figée peut être choisie si :

- N est petit,
- λ ou T est petit,
- γ est grand.

La duplication figée peut être choisie pour un système petit, facile à réparer, fréquemment maintenu.

La triplication dynamique n'est jamais une bonne solution : elle est coûteuse et peu disponible. Il sera pourtant parfois indispensable de la choisir si les contraintes de temps sont importantes ou si on ne veut pas déterminer des points de reprise dans le programme.

IV - 3 DEGRADATION DANS UNE HIERARCHIE DE SOUS SYSTEMES MODULAIRES (26)

IV - 3 a DESCRIPTION DU SYSTEME

Chaque sous système exécute une partie d'une tâche et une tâche doit être exécutée par tous les sous-systèmes successivement. Un tel système est équivalent à la structure suivante :

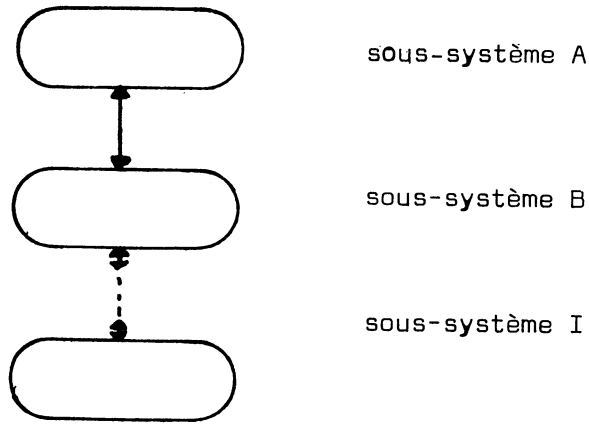


Fig. II - 17

Les sous-systèmes A, B... I sont composés de N_A, N_B..., N_I modules. Le nombre de modules dans chaque système est calculé par le concepteur de telle sorte que le système, en l'absence de panne, ait la puissance souhaitée. Un tel système est dit équilibré.

IV - 3 b PUISSANCE D'UN TEL SYSTEME

Comme le système est équilibré, la puissance normalisée $P(T)$ du système en présence de pannes est égale au minimum des puissances normalisées $P_i(T)$ des sous-systèmes :

$$P(T) = \min(P_i(T))$$

Donc la μ fiabilité $R_T(\mu)$ du système peut être calculée à partir des μ fiabilités $R_T^i(\mu)$ des sous-systèmes.

$$R_T(\mu) = \prod_i R_T^i(\mu) = \prod_i R_{Ni,T}(\mu.Ni)$$

si N_i est le nombre de modules du sous-système i,

$$A_T(\mu) = \prod_i A_T^i(\mu) = \prod_i A_{Ni\ T}(\mu.Ni)$$

IV - 3 c EXEMPLE : système non redondant.

Considérons un système composé de 3 sous-systèmes A, B, C où $N_A = 2$, $N_B = 3$, $N_C = 6$ si le système est équilibré.

La figure 6 montre les courbes de $R_T(\mu)$ si tous les modules du système ont un taux de panne tel que $\lambda T = 10^{-3}$.

Courbe a : le système est équilibré,

Courbe b : on rajoute un module dans chaque sous-système pour obtenir une meilleure fiabilité : $N'_A = 3$, $N'_B = 4$, $N'_C = 7$.

On voit qu'au prix d'un module supplémentaire dans chaque sous-système, le système a une puissance égale à 1 avec une probabilité 0.99990 dans ce cas.

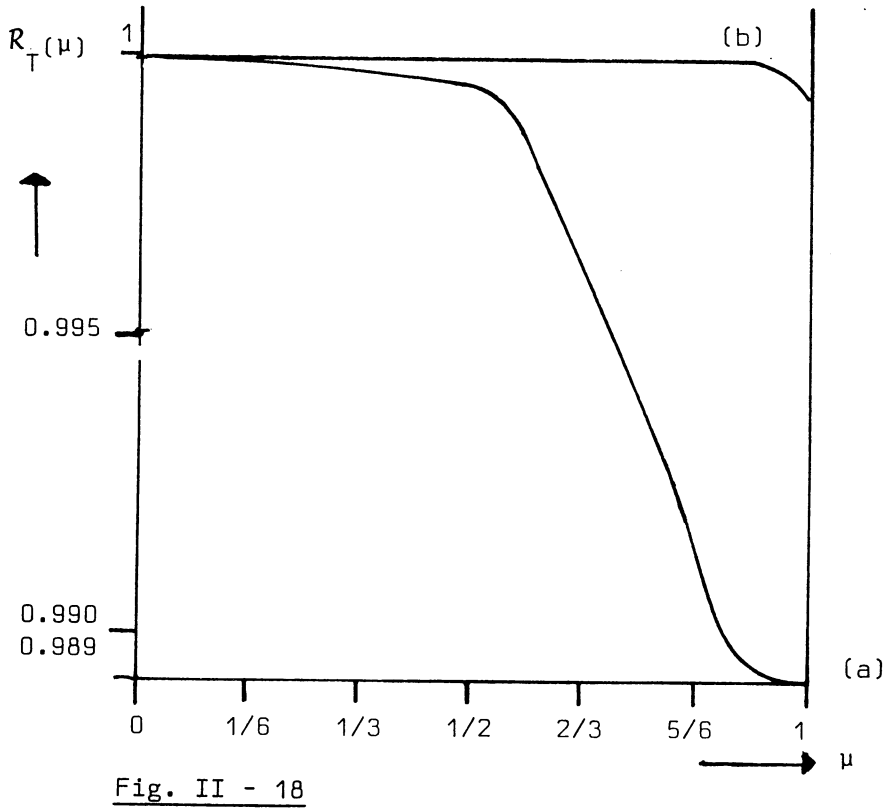


Fig. II - 18

La figure II - 19 montre les courbes de $R_T(\mu)$, les sous-systèmes ayant de modules de fiabilités différentes.

		A $N_A=2$	B $N_B=3$	C $N_C=6$
courbe à	λT =	10^{-3}	10^{-3}	10^{-3}
courbe b	λT =	10^{-2}	10^{-3}	10^{-3}
courbe c	λT =	10^{-3}	10^{-2}	10^{-3}
courbe d	λT =	10^{-3}	10^{-3}	10^{-2}

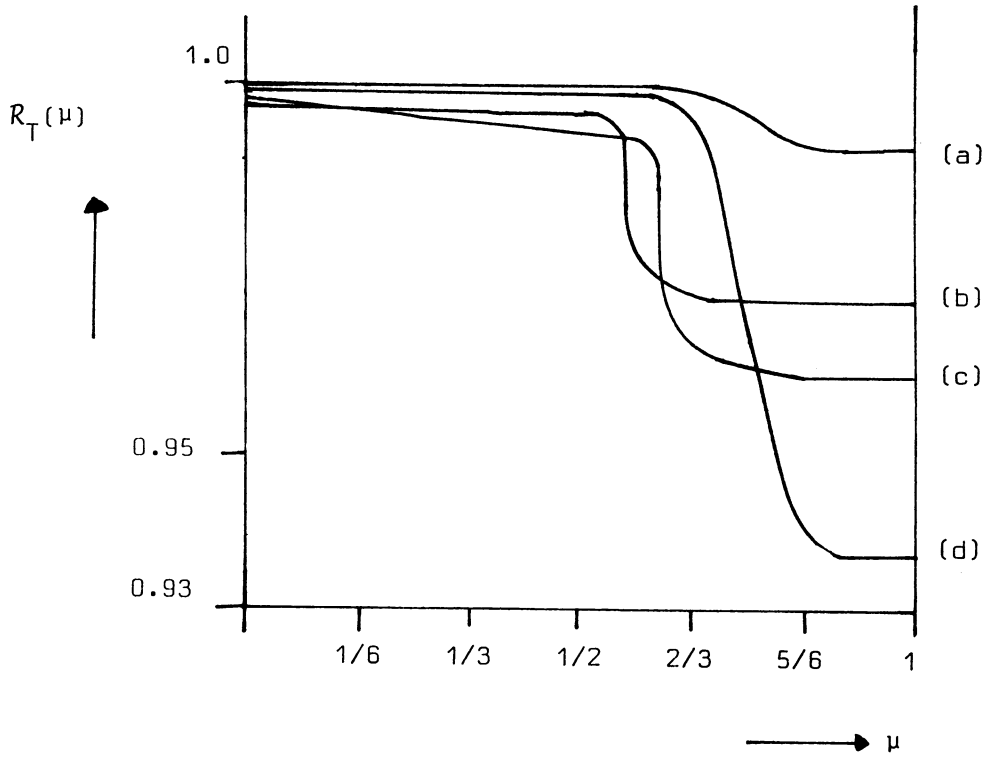


Figure II-19

La probabilité que la puissance du système soit 1 est :

$$R_T(1) = \prod_i ({}^iR(T))^{Ni}$$

où ${}^iR$ est la fiabilité des modules du sous-système i

${}^iR \approx 1 - \lambda iT$

où λi est le taux de pannes des modules du sous-système i.

$\lambda i\, T$ étant de l'ordre de 10^{-3} à 10^{-2} , l'approximation est possible.

$$R_T(1) \approx \prod_i (1 - \lambda i\, T)^{Ni} \approx \prod_i (1 - Ni\ \lambda i\ T) \approx 1 - \sum_i Ni\ \lambda i\ T \ .$$

En conséquence,

$$\frac{\delta\, R_T(1)}{\delta\, \lambda i\ T} = - Ni$$

Dans l'exemple,

$$\Delta\ \lambda i\ T = 10^{-2} - 10^{-3} \approx 10^{-2}$$

$\Delta\ R_T(1) = 0.02$ si $i=A$

$= 0.03$ si $i = B$

$= 0.06$ si $i = C$

La puissance moyenne du système est, avec une approximation du premier ordre (un seul module en panne) :

$$P_{moy}(T) = \sum_i \left[\frac{Ni - 1}{Ni} \cdot C^1_{Ni} \cdot ({}^iR)^{Ni-1} \cdot (1 - {}^iR)\right] + R_T(1) \ .$$

$({}^iR)^{Ni-1} \cdot (1 - {}^iR) \approx 1 - {}^iR$ puisque $(1 - \varepsilon).(\varepsilon) = \varepsilon - \varepsilon^2 \approx \varepsilon$

$$P_{moy}(T) = \sum_i (Ni - 1)\ \lambda i\ T + 1 - \sum_i Ni \cdot \lambda i \cdot T = 1 - \sum_i \lambda i\ T$$

$$\Delta\ P_{moy}(T) = - \Delta\ \lambda i\ T$$

La puissance moyenne est peu sensible au nombre de modules dans les sous-systèmes.

IV - 3 e CONCLUSION

Le concepteur choisira les modules les plus fiables pour le sous-système contenant la plus de modules, ce qui accoît considérablement la probabilité que la puissance soit 1, et modifie peu la puissance moyenne (indépendante du nombre de modules dans le sous-système considéré).

IV - 4 NOMBRE DE MODULES NECESSAIRES POUR ASSURER UNE PUISSANCE MINIMALE FIXEE.

IV - 4 a PUISSANCE MINIMALE

Pour chaque sous-système modulaire d'un système, une puissance minimale est définie. Cette puissance est assurée si il y a N_i° modules dans le sous-système i. Connaissant λi et γi, respectivement taux de pannes et taux de réparation des modules du sous-système i, on peut déterminer le nombre Ni (T,Q) $\geq N_i^\circ$ de modules nécessaires pour avoir au moins N_i° modules en état de marche à l'instant T avec la probabilité Q. (Q est très proche de 1).

IV - 4 b EVALUATION DE Ni (T,Q)

Le problème est donc de trouver pour chaque sous-système i, le nombre $N_i(T,Q)$ tel que $R_{Ni,T}(N_i^\circ) \geq Q$, T étant le temps de mission.

Considérons la famille de courbes de la figure II - 20. Les différentes courbes représentent $R_{N,T}$ (n) pour $N = N_i^\circ$, $N_i^\circ + 1$, $N_i^\circ + 2$.

On trace les droites :

D1 : $n = N_i^\circ$

D2 : $Rn = Q$

La solution est donnée par la première courbe dont l'intersection avec D1 est d'abscisse supérieure à Q.

Dans ce cas $Ni = N_i^\circ + 1$.

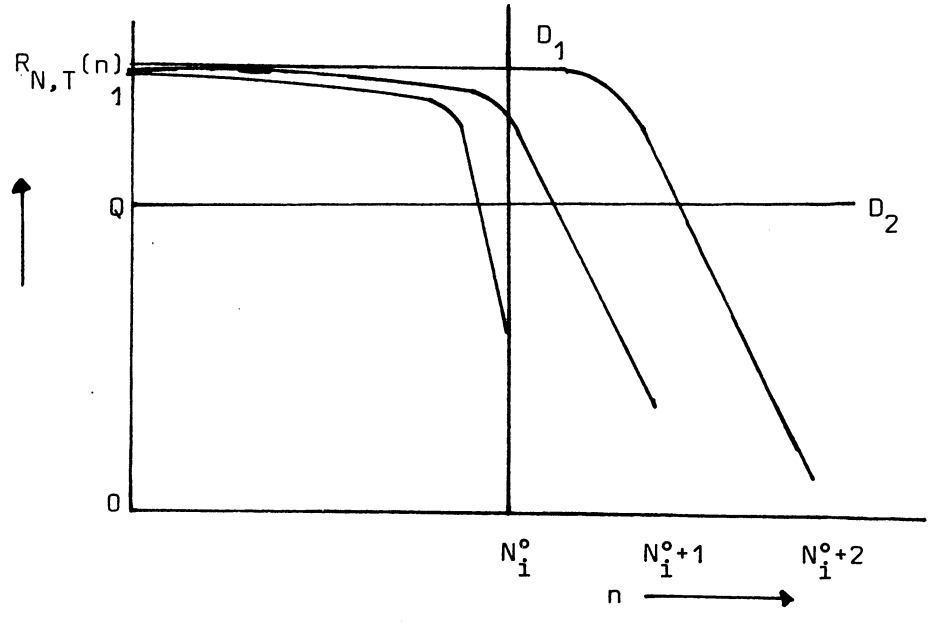


Figure II-20

Exemple :

Considérons un sous-système modulaire avec : $N_i^{\circ} = 8$, $\lambda T = 2.10^{-2}$, $Q = 0.99$

La figure II - 21 montre les courbes de $R_{8,T}(n)$, $R_{9,T}(n)$, $R_{10,T}(n)$. On trouve ici Ni = 10.

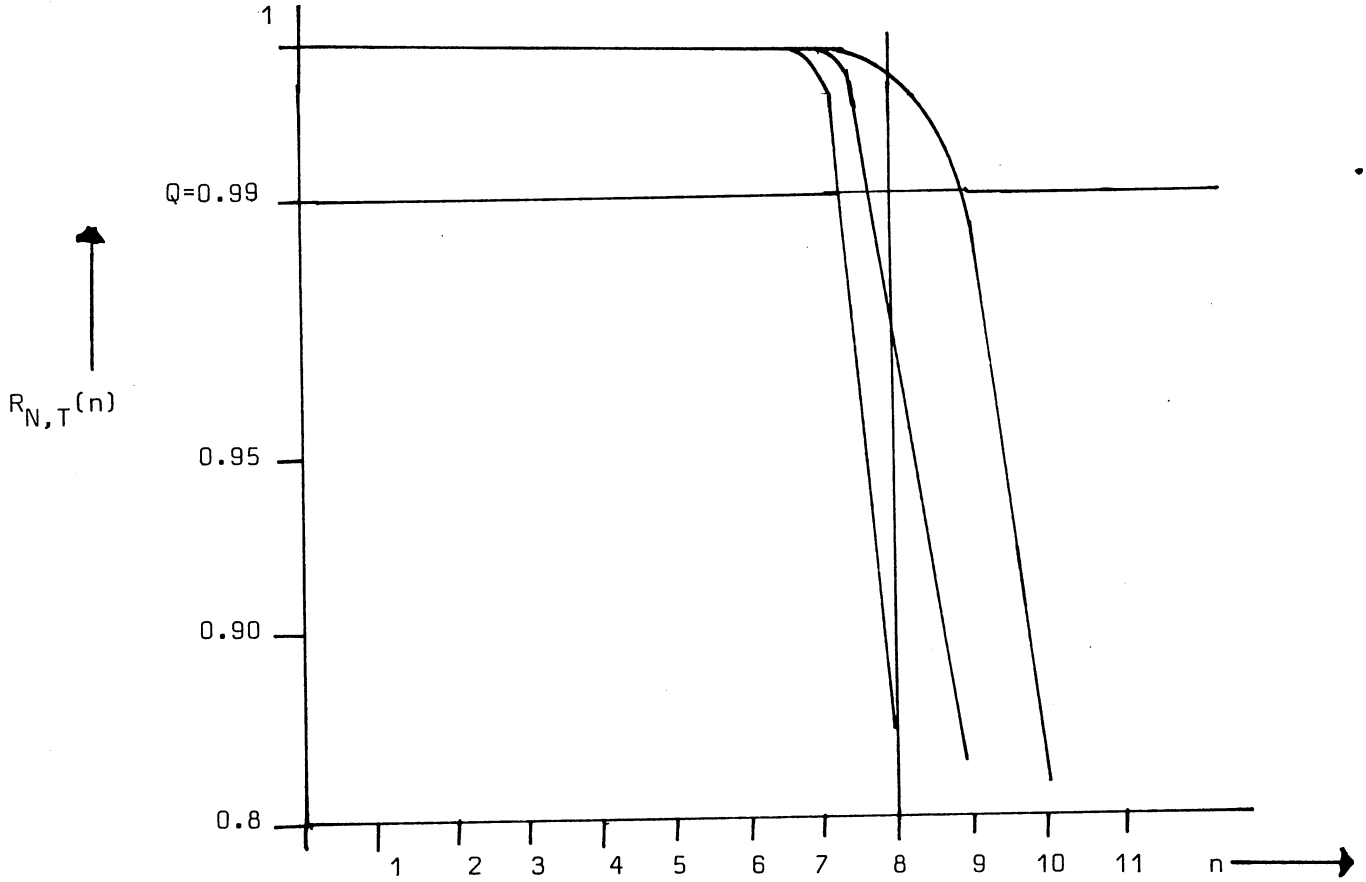


Figure II.21

V - CONCLUSION

Les définitions données permettent d'évaluer les grandeurs interessantes de fiabilité et disponibilité dans un système réparti.

Cette évaluation sera étendue par la suite à des systèmes réalisant plusieurs fonctions ; on évaluera alors la disponibilité de chacune de ces fonctions et non plus la disponibilité du système complet. Cela permettra de faire intervenir une pondération, suivant l'importance de la fonction.

CHAPITRE III

DETECTION CONTINUE EN COURS DE FONCTIONNEMENT.

CONTAMINATION ENTRE PROCESSUS.

I - INTRODUCTION

Après avoir étudié l'aspect architectural de la sûreté de fonctionnement d'un système réparti, nous nous intéressons à l'aspect logiciel.

Un système réparti sera considéré par la suite comme un ensemble de couple (processeurs, processus). Un processus est l'ensemble des algorithmes implemen tés sur le processeur qui est associé au processus : il existe une bijection entre l'ensemble P des processus et l'ensemble M des processeurs. Un processus $p \in P$ est une entité logicielle, un processeur $m \in M$ est une entité matérielle un couple (m,p) peut être soit un module, soit un sous système (voir chapitre

Les couples (processeurs, processus) peuvent être interconnectés de diverses façons (mémoire partagée, boîte aux lettres, envois de messages...). Le matériel d'interconnection, partagé par les processeurs, n'appartient à aucun d'entre eux.

Un système S peut donc être représenté par le schéma suivant :

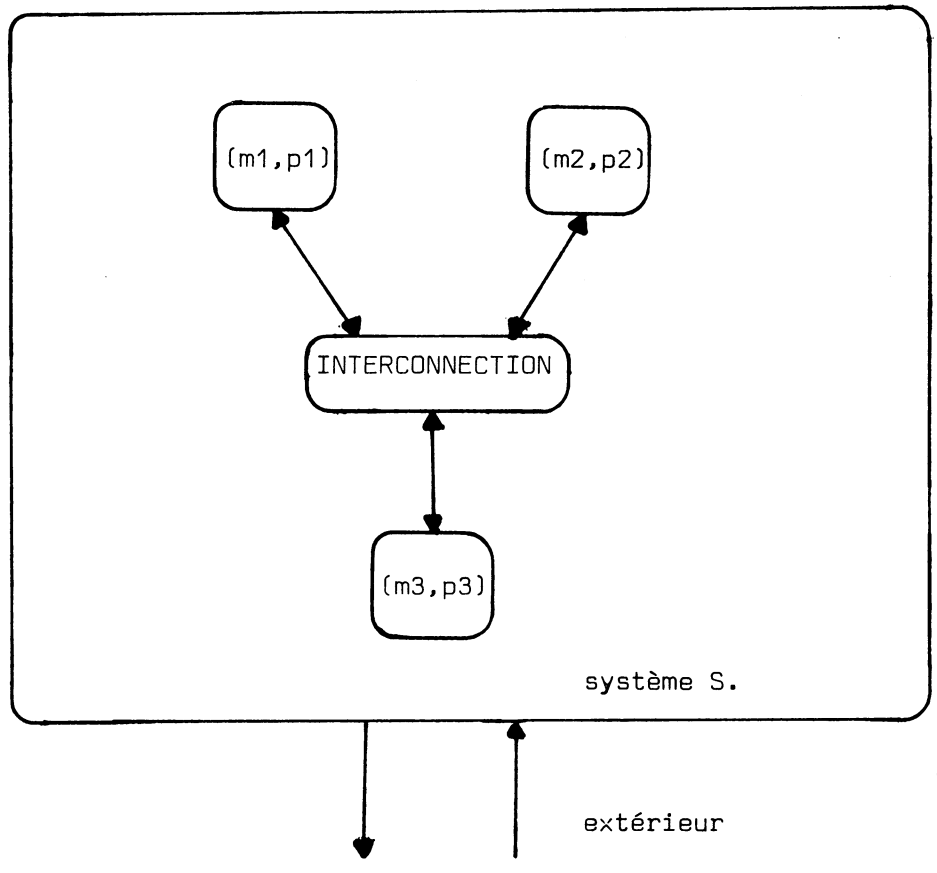


Fig. III - 1

Le système S peut être considéré comme un seul couple processeur-processus à un niveau plus macroscopique; et inversement chacun des couples (mi, pi) du système S peut être considéré comme un système complet et décomposé en plusieurs couples. (Cela suppose la décomposition du processus Pi en sous-processus P_i^j, telle que chaque sous-processus P_i^j soit exécuté par un sous-processeur M_i^j indépendant).

La sûreté de fonctionnement et la reconfiguration d'un système réparti modulaire, nécessite une détection des pannes et des erreurs extrêmement rapide et efficace. Cette détection doit donc se faire au cours du fonctionnement normal. Elle peut être continue ou discontinue.

Détection continue :

Elle se fait au cours de l'exécution normale d'un processus. Elle peut être assurée par la redondance matérielle. Nous nous intéressons ici à la détection assurée par redondance du logiciel, qui consiste à vérifier le bon fonctionnement d'un processeur à travers l'éxécution correcte du processus. La détection continue détecte les erreurs (et non les pannes).

Détection discontinue :

Soit on utilise des temps d'inactivité du processeur, soit on interrompt durant une courte période l'activité du processeur pour lui faire exécuter un programme de test qui détecte les pannes.

La remise en état des informations logicielles nécessite un moyen d'étudier la propagation des erreurs dans un processus, ou contamination.

Pour étudier la détection et la contamination, nous proposons en II une modélisation adéquate des processus; sur ce modèle, les principes de détection continue sont énoncés en III et un algorithme d'étude de la contamination des données entre processus en IV. La détection discontinue sera étudiée au chapitre IV de cette étude.

II - MODELISATION DES PROCESSUS

Parmi les représentations possibles d'un processus, nous en choisirons une particulièrement adaptée à cette étude : le réseau de Pétri interprété et étiqueté. Cette modélisation présente en outre l'avantage de pouvoir être décrite en langage MAS et simulée (14).Mais les méthodes employées peuvent être trans-

posées à tout autre modèle.

II - 1 RAPPEL SUR LES RESEAUX DE PETRI (15) , (16).

Un réseau de Pétri est un graphe orienté biparti, représenté par un triplet (P,T,V) où :

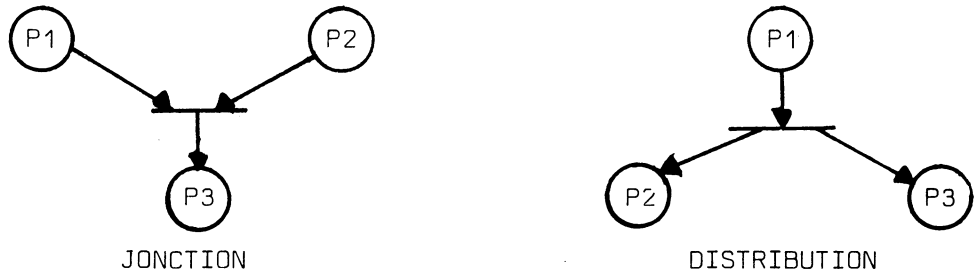
- P est un ensemble fini de places,
- T est un ensemble fini de transitions,
- V est un ensemble fini d'arcs ; un arc relie une place à une transition ou une transition à une place.

Un marquage M détermine l'état du réseau de Pétri. C'est une bijection de P et de l'ensemble des entiers positifs ou nuls. A chaque place, un marquage M associe un nombre : nombre de jetons. Ces jetons se déplacent dans le réseau avec les règles suivantes :

- une transition est validée si toutes ses places antécédentes possèdent au moins un jeton,
- une transition validée est alors mise à feu : un jeton est supprimé dans chacune de ses places antécédentes, et chacune des places successeurs reçoit un jeton supplémentaire.

Un marquage Mj est accessible à partir d'un marquage Mi, s'il existe une séquence de mise à feu qui, à partir du marquage Mi, conduit au marquage Mj . Un réseau de Pétri possède un marquage initial Mo. On distingue deux types de noeuds :

Noeud et



JONCTION DISTRIBUTION

Fig III - 2

jonction : p3 reçoit un jeton si il y a un jeton dans p1 et p2 et si t est mise à feu.

distribution : p2 et p3 reçoivent un jeton si il y a un jeton dans p1 et si t est mise à feu

Noeud ou

ATTRIBUTION SELECTION

Fig. III - 3

attribution : p3 reçoit un jeton s'il y a un jeton dans p1 et t1 est mise à feu, ou s'il y a un jeton dans p2 et t2 est mise à feu.

sélection : s'il y a un jeton dans p1, p2 reçoit un jeton si t1 est mise à feu, ou p3 reçoit un jeton si t2 est mise à feu.

II - 2 RESEAUX DE PETRI AYANT DES PROPRIETES PARTICULIERES

Un réseau est borné pour un marquage initial Mo, si pour tout marquage accessible à partir de Mo, aucune place de P ne contient plus d'un nombre fixé entier positif n de jetons.

Un réseau est sauf pour Mo si et seulement si pour tout marquage accessible à partir de Mo, aucune place de P ne contient plus d'un jeton.

Un réseau est vivant pour Mo si et seulement si pour tout marquage Mi accessible à partir de Mo et pour toute transition t ϵ T, il existe un marquage Mj accessible à partir de Mi qui valide t.

Un réseau est propre pour Mo si pour tout marquage Mi accessible à partir de Mo, Mo est accessible à partir de Mi.

II - 3 RESEAUX DE PETRI INTERPRETES ET ETIQUETES

Réseau de Pétri interprété : une place est une phase de l'exécution d'un processus. A chaque place est associée une procédure qui est l'ensemble des actions que le processus doit exécuter à cette phase de l'exécution. La procédure est activée par la présence d'un jeton dans cette place.

La transition successeur de la place est validée quand l'exécution de la procédure est terminée.

Réseau de Pétri déterminé : dans deux places exécutables en parallèle, il n'existe pas de conflits sur les ressources matérielles ou logicielles, nécessaires à l'éxécution des procédures.

Réseau de Pétri interprété et étiqueté : à chaque transition est associé un prédicat. Une transition validée est mise à feu si le prédicat est vrai. La valeur du prédicat dépend soit des variables internes du processus, soit elle est conditionnée par un autre processus ou le monde extérieur.

Un réseau de Pétri étiqueté est déterministe si les prédicats portés par les transitions successeurs d'une même place sont exclusifs deux à deux. Une seule transition successeur d'une place p peut être mise à feu à un instant donné. Il n'y a donc pas de choix possible.

Un Réseau de Petri est préstructuré si :

- il est connexe
- il possède une place initiale I, une place finale F, et la place F possède une transition successeur unique to (transition de retour) qui est la transition antécédente unique de I
- il est sauf et vivant pour le marquage initial Mo : un seul jeton dans le réseau, à la place I.

On peut démontrer :

Proposition 1

Un réseau préstructuré est propre et il existe un et un seul marquage successeur de Mo tel qu'il y ait un jeton en I et ce marquage est Mo lui-même.

Démonstration : Le réseau étant vivant pour Mo, il existe une séquence de mise à feu qui valide to. Appelons M le marquage résultant de la mise à feu de to. On a :

$$M \geq Mo$$

Supposons que $M > Mo$: il existe une place p différente de I ayant un jeton. On peut mettre à feu à partir de M la même séquence . Il en résultera un deuxième jeton dans p : le réseau ne sera pas sauf. Donc $M = Mo$.

Proposition 2

Tout point du réseau appartient à un chemin allant de I à F.

Démonstration : Par exemple, prenons une place p telle que :

1er cas : il n'y a pas de chemin de I à p.

Le réseau étant connexe cette place a au moins une transition antécédente ou une transition successeur. Dans tous les cas cette transition ne pourra être validée à partir de Mo puisqu'il n'y a pas de chemin de I à p. Le réseau n'est pas vivant.

2e cas : il n'y a pas de chemin de p à F.

Il existe un marquage M accessible à partir de Mo qui marque p (d'après le 1er cas, p a une transition antécédente qui peut être validée à partir de Mo). Il existe un marquage M' accessible à partir de M qui valide to et tel que le jeton de p ne soit pas ôté (en effet, le tir d'une transition successeur de p ne modifie pas le cheminement des jetons vers F puisqu'il n'y a pas de chemin de p à F). Après mise à feu de to, on obtient un marquage M" > Mo, ce qui est contraire à la proposition 1.

Corrolaire : Un réseau préstructuré est fortement connexe.

Un écoulement dans un réseau préstructuré (17) est un ensemble de places et de transitions tel que, après suppression de la transition de retour (reliant F à I) :

- il contient les phases I et F,
- si une transition appartient à l'écoulement, toute place antécédente ou successeur de cette transition appartient à l'écoulement,
- pour toute place de l'écoulement il existe un chemin, passant par cette place, venant de I et allant à F, et appartenant à l'écoulement.

Un écoulement est une des activations fonctionnelles possibles du processus.

Exemple:

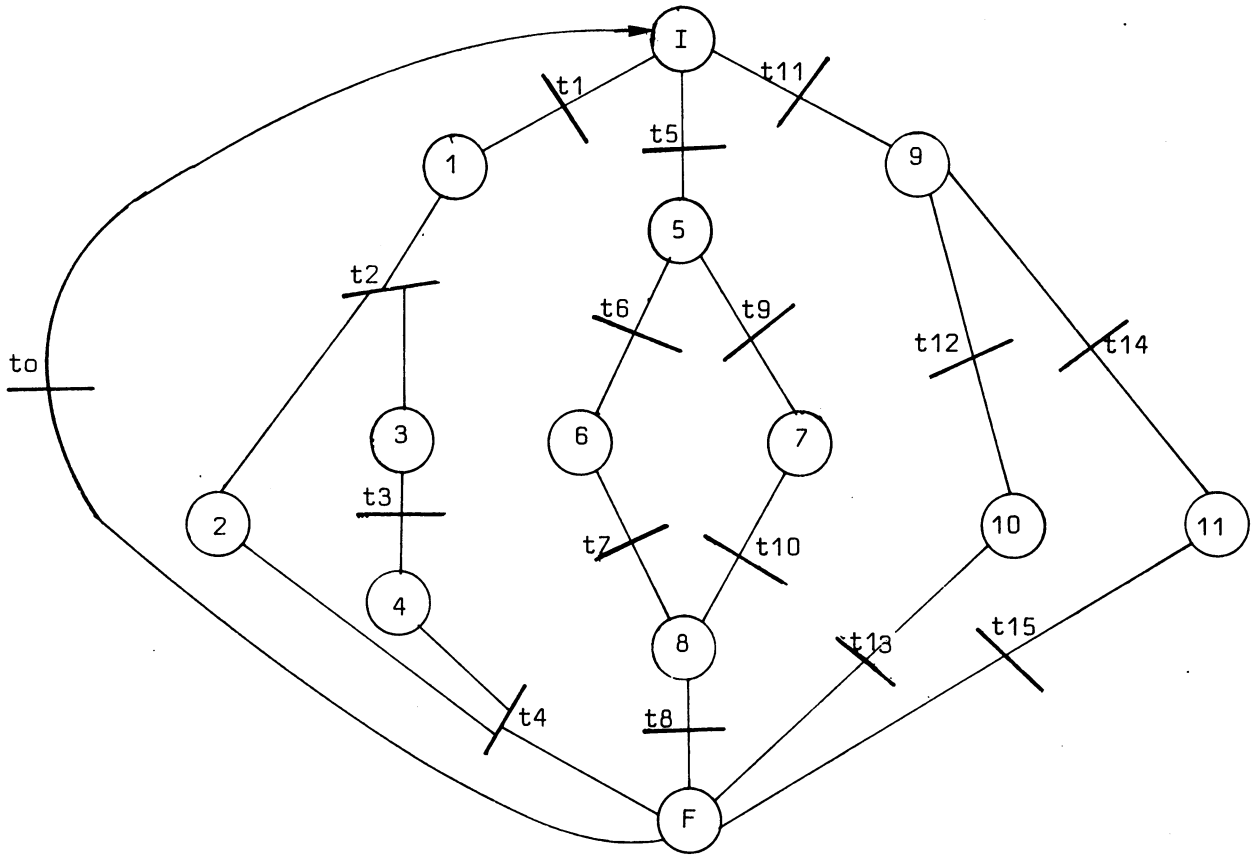


Figure III.4

La liste des écoulements est :

E1 = {I,t1,1,t2,2,3,t3,4,t4,F}
E2 = {I,t5,5,t6,6,t7,8,t8,F}
E3 = {I,t5,5,t9,7,t10,8,t8,F}
E4 = {I,t11,9,t12,10,t13,F}
E5 = {I,t11,9,t14,11,t15,F}

Sous processus d'un réseau pré- structuré : étant donné la place initiale I unique, la première divergence induit une partition en sous-ensembles d'écoulements ou sous processus. Ces sous processus ont en général une interprétation fonctionnelle de haut niveau.

Exemple : dans la figure III - 4, il y a trois sous-processus :

SP1 = {E1}
SP2 = {E2,E3}
SP3 = {E4,E5}

II - 4 SYNCHRONISATION ENTRE PROCESSUS REPRESENTES PAR DES RESEAUX PRE-STRUCTURES, INTERPRETES ET ETIQUETES.

Cette étude est centrée sur l'étude des processus et de leur protection fonctionnelle, non sur la synchronisation entre processus, qui est hors de notre propos. Aussi, chaque processus sera représenté par un réseau pré-structuré, déconnecté des autres processes.

La synchronisation sera faite par des variables partagées , définies comme suit : soit deux processus P_1 et P_2 (fig III - 5 - a) formant un processus qui n'est pas structuré. Supposons que l'exécution de la procédure associée à Pk $\in P_1$ ne puisse être initialisée que si la procedure Pj $\in P_2$ soit terminée (processus P_1 synchronisé sur le processus P_2) ; dans ce cas, on définit une variable de synchronisation t ; cette variable sera à 1 à la fin de la procédure de Pj et figurera dans le prédicat de la transition de Pi à Pk. Le réseau est alors transformé en deux sous-réseaux, synchronisés par la variable t,ainsi que l'indique la figure III - 5 - b.

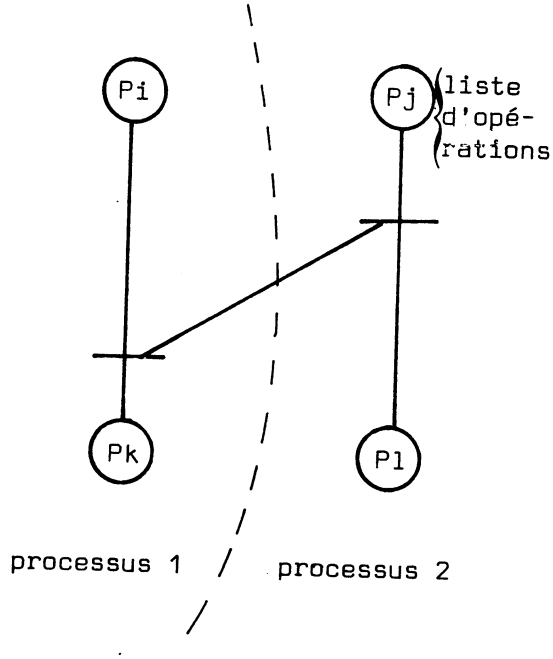


Figure III - 5 - a

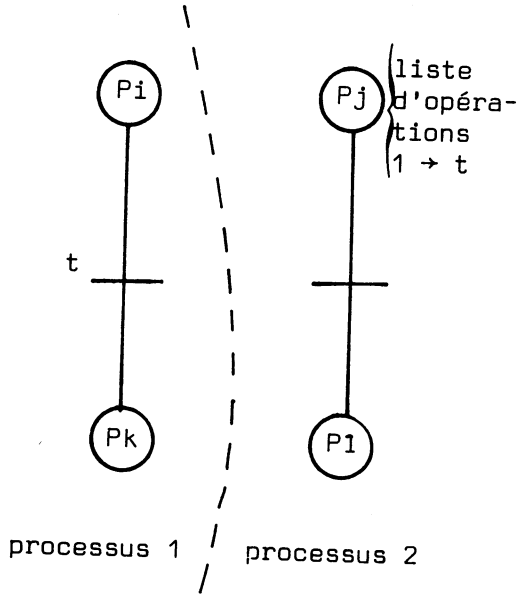


Figure III - 5 - b

III - DETECTION CONTINUE EN COURS DE FONCTIONNEMENT (20)

Un processus est représenté par un réseau de Pétri pré-structuré, interprété et étiqueté.

III - 1 CLASSIFICATION DES PLACES D'UN PROCESSUS.

On distingue certains types de places suivant leur interprétation, c'est à dire en considérant les opérations effectuées par les procédures associées. On appelera désormais phase une place interprétée par sa procédure;

Les phases sont classifiées suivant leur propriétés au point de vue détection et sûreté de fonctionnement.

- phase obligatoire d'un processus :
  c'est une phase qui appartient à tous les écoulements du processus ; les phases I et F sont des phases obligatoires.

- phase caractéristique d'un écoulement :
  c'est une phase d'un processus qui appartient à un et un seul écoulement de ce processus.

- phase observable d'un processus :
  c'est une phase générant un échange entre processus, ou positionnant une variable de synchronisation. Le processus est alors observé par un autre processus.

- phase sûre d'un processus :

  l'exécution de cette phase est garantie correcte parce qu'elle s'exécute sur un matériel autotesté. On considère alors que si les entrées d'une telle phase sont justes, ses sorties aussi,ou qu'un signal d'erreur est émis.

- phase partiellement autotestée :
  l'exécution de cette phase est protégée contre un certain nombre de pannes ; cette phase est exécutée par un matériel partiellement autotesté.

- phase dangereuse d'un système :
  c'est une phase qui commande la délivrance d'une valeur au monde extérieur. Une phase dangereuse est une phase observable par l'extérieur ; l'inverse n'est pas vrai.

- phase de vérification d'un processus : c'est une phase qui n'est pas nécessaire fonctionnellement pour exécuter le processus mais qui exécute une vérification fonctionnelle ou matérielle pour détecter la présence d'erreurs ou de pannes.
- phase de contamination d'un processus : c'est une phase qui commande la délivrance d'une valeur à un autre processus. Une phase de contamination est une phase observable du processus ; l'inverse n'est pas vrai.
- phase de réception d'un processus : c'est une phase qui contrôle la réception d'une valeur du monde extérieur ou d'un autre processus.

Exemple :

Soit un processus communiquant avec deux autres processus P1 et P3

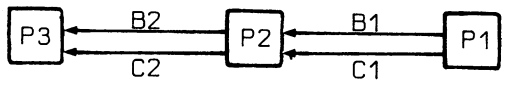


Fig. III - 6

Il reçoit de P1 les valeurs B1 et C1, quand la variable de synchronisation t vaut 1. Il effectue un certain traitement et envoie au processus P3 les valeurs B2 et C2. La modélisation du processus est la suivante :

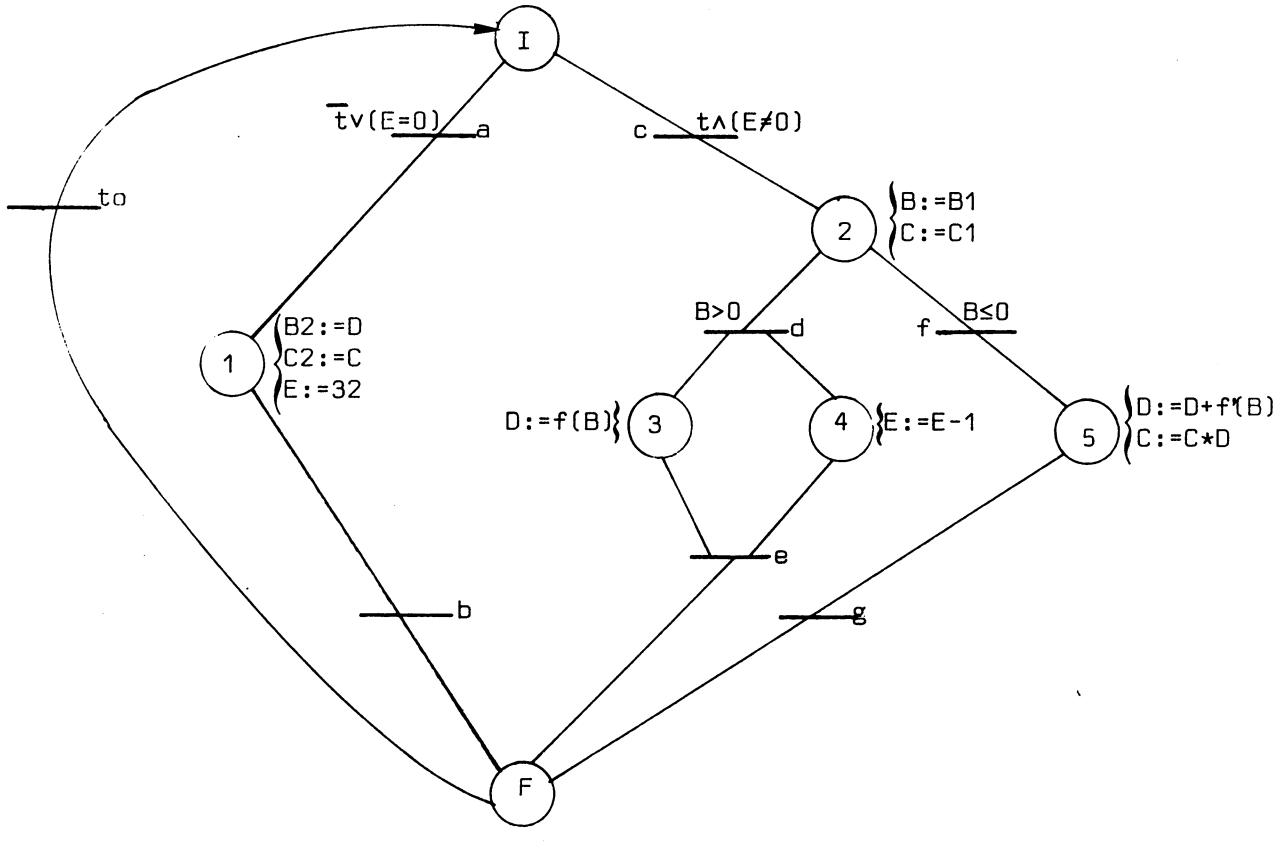


Fig. III - 7

- les écoulements sont :

E1 = {I,a,1,b,F}
E2 = {I,c,2,d,3,4,e,F}
E3 = {I,c,2,f,5,g,F}

- les sous-processus sont :

SP1 = {E1} communication avec P3
SP2 = {E2,E3} communication avec P1

- les phases obligatoires du processus sont I et F.
- les phases caractéristiques des écoulements sont 1 pour E1, 3 et 4 pour E2, 5 pour E3.
- la phase 1 est une phase de contamination du processus P2 considéré : elle peut envoyer des valeurs fausses (B2 et C2) au processus P3. C'est aussi une phase observable par le processus P3.
- la phase 2 est une phase de réception du processus P2 qui peut recevoir une valeur fausse (B1 et C1) envoyée par le processus P1.

III - 2 ANALYSE DES ERREURS DANS UN PROCESSUS.

Une panne survenant dans le matériel sur lequel s'exécute un processus, peut avoir deux manifestations :

a) séquencement erroné dans le processus ; il s'agit soit d'un bouclage, soit d'un passage illégal d'une phase à une autre
b) erreur sur les données traitées, le séquencement restant correct. Le test du processus en cours de fonctionnement visera donc à détecter ces deux types d'erreur.

III - 3 TEST DU SEQUENCEMENT DU PROCESSUS.

Ce test tend à détecter les erreurs de séquencement, indépendemment des données traitées.

a) <u>test de bouclage</u> :

Ce test consiste à repérer le passage du processus dans une ou plusieurs phases du processus. Connaissant le fonctionnement du processus, il faut évaluer la borne supérieure T du temps qui sépare deux passages dans cette phase, en l'absence de pannes. Le dépassement de cette borne T peut être détectée soit par

le matériel support du processus lui-même, en créant une phase de vérification exécutée sur une partie hard-core de ce matériel (compteur), soit par un autre matériel, si la phase choisie est une phase observable. On a dans le deuxième cas un contrôle de type Watch-dog.

Exemple :

Pour le processus de la figure III - 7, deux solutions sont possibles :

- choisir T égal au temps maximum d'éxécution d'un écoulement et implémenter une horloge H qui est remise à 0 à chaque passage dans la phase initiale I, et qui, quand H = T délivre un signal d'erreur.
- si la valeur de B est positive et négative ou nulle, avec la même probabilité, on passe dans la phase 1 en moyenne tous les 64 T. On peut donc implémenter dans P3 un contrôle de style watch-dog, qui signale une erreur si P2 n'a pas communiqué avec P3 au bout de 100 T.

b) test de passage illégal d'un sous-processus à un autre :

- on choisit pour chaque écoulement une phase caractéristique s'il en existe, sinon on en crée une. On vérifie qu'au cours de l'exécution du processus on passe par une phase caractéristique et une seule entre deux passages à la phase initiale I.
- si un découpage en sous-processus a été réalisé, on mémorise l'entrée dans un sous-processus ; au passage dans une phase caractéristique, on vérifie l'appartenance de cette phase au sous-processus en cours d'exécution.

Dans les deux cas, les phases caractéristiques doivent être rendues observables ou doivent être des phases de vérification.

Exemple :

Pour le processus de la figure III - 7, le test de passage d'un sous-processus à un autre sera réalisé à l'aide des phases caractéristiques. Un compteur γ est incrémenté à chaque passage dans une phase caractéristique. Il est remis à 0 par la phase initiale I. La valeur de γ est testée en F et avant la phase 1 qui est une phase de contamination. Une phase "erreur" est créee. Le prédicat R est mis à 1 après traitement de l'erreur, pour permettre de continuer l'exécution. On obtient le processus suivant :

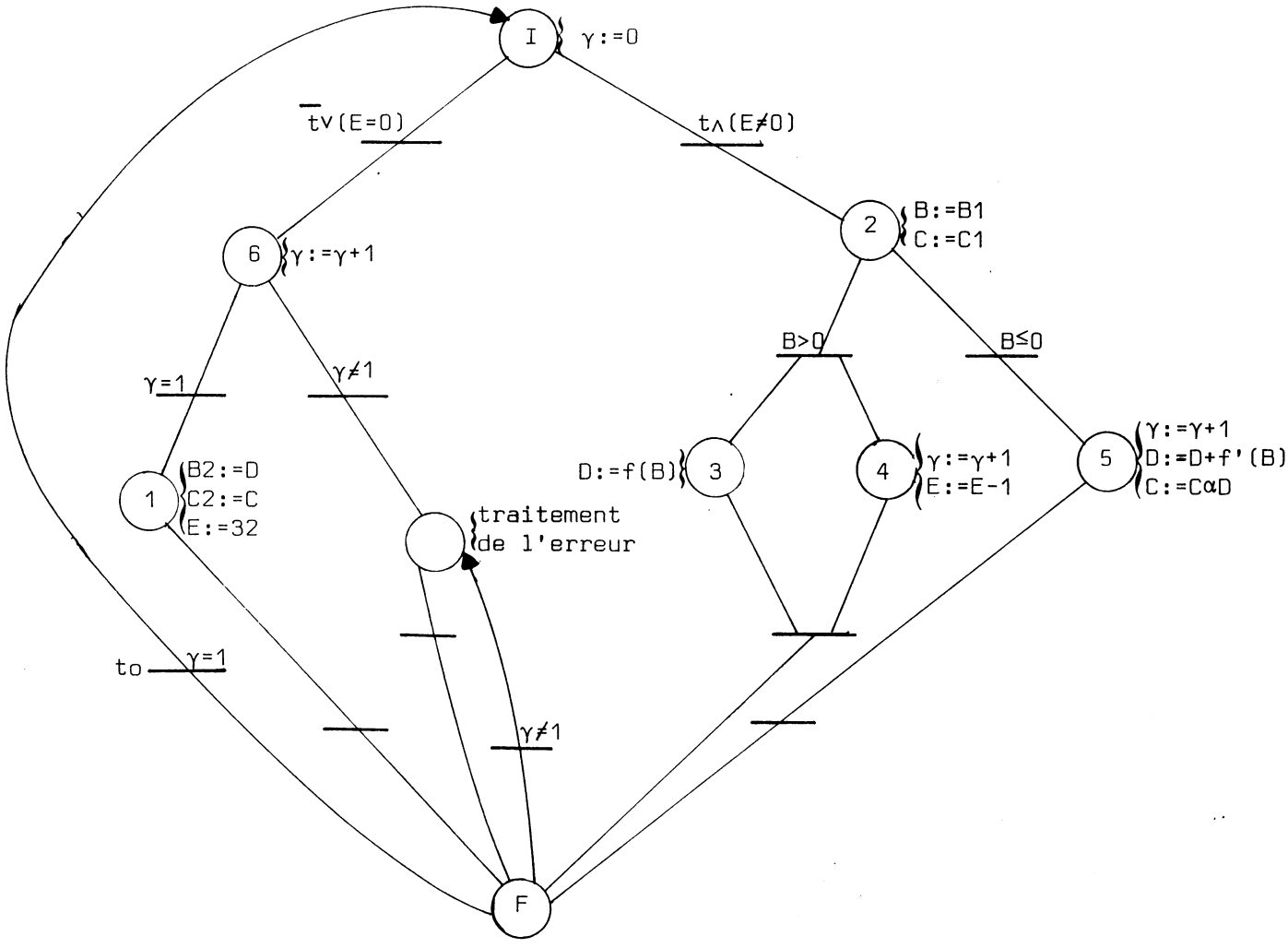


Fig III - 8

III - 4 TEST DES DONNEES .

Il est difficile de vérifier que les valeurs calculées, etc... ne sont pas erronées, sans utiliser de redondance. Deux moyens se présentent cependant : le test de vraisemblance et le test d'adressage.

a) test de vraisemblance :

Il consiste à vérifier que les valeurs calculées sont comprises entre deux bornes dites bornes de vraisemblance. Il faut donc pouvoir fixer ces bornes, ce qui peut être fait en règle générale au moins pour les systèmes de contrôle de processus.

b) test d'adressage, au moment de l'accès à une mémoire partagée:

Si une partie de la mémoire est allouée de façon dynamique à un processus

et un seul, à chaque tentative d'accès on vérifiera que l'adresse demandée est bien dans la zone réservée au processus. Cela peut se faire à l'aide d'une des techniques bien connues de protection de l'information (base limite, liste d'accès liste des droits, clés et verrous).

Les vérifications de vraisemblance et d'adressage seront effectuées par des phases fictives de vérification, implementées dans le processus.

Exemple :

On ajoute au processus de la figure III - 8, un contrôle de vraisemblance sur la valeur D, qui doit être comprise entre les valeurs X et Y. On obtient le processus suivant :

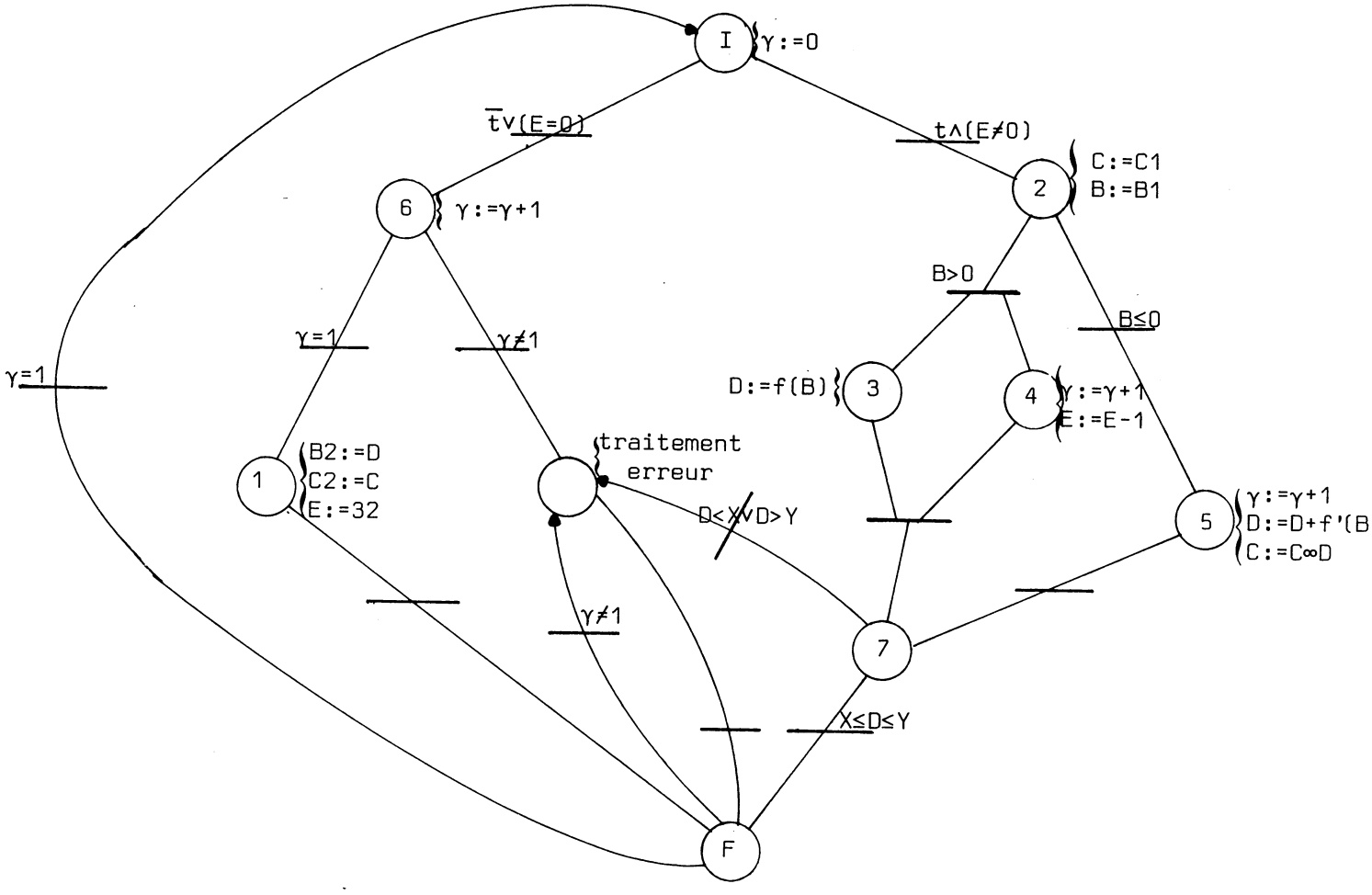


Figure III - 9

III - 5 ISOLATION ENTRE PROCESSUS ET SYSTEME A HAUTE SECURITE

Un processus erroné peut contaminer un autre processus en lui communiquant des valeurs fausses au cours d'un échange. Les phases où il peut se produire une telle contamination ont été définies comme des phases de contamination. La contamination rend difficile la reprise du fonctionnement et impossible la

localisation (même grossière) de la panne. Il est donc important d'isoler-les processus au point de vue des erreurs, c'est-à-dire d'interdire la communication d'une valeur fausse à un autre processus. L'isolation est assurée en plaçant, si possible, une phase fictive de vérification avant chaque phase de contamination du processus.

Un système est sûr s'il ne communique pas de valeurs erronées au monde extérieur. Il faudra donc protéger chaque phase dangereuse du système c'est-à dire les phases où une valeur est communiquée au monde extérieur (processus contrôlé par le système par exemple). On placera avant chaque phase dangereuse une phase de vérification aussi exhaustive que possible, qui peut aller jusqu'à un test exhaustif du matériel utilisé par l'exécution de l'écoulement.

Dans certains cas, si une phase dangereuse appartient à un nombre restreint d'écoulements et que la valeur qu'elle communique à l'extérieur est particulièrement critique, on peut protéger les écoulements conduisant à cette phase en rendant sûres toutes les phases de ces écoulements. Ceci amène à implementer de la redondance sur le matériel utilisé par ces phases.

III - 6 EVALUATION DE L'EFFICACITE.

L'efficacité de la détection en cours de fonctionnement peut être mesurée par trois paramètres : le taux de pannes détectées $\mu 1$, le taux d'isolation $\mu 2$, et le taux de dégradation des performances $\mu 3$.

a) taux de pannes détectées $\mu 1$:

L'efficacité en taux de pannes détectées ne peut être évaluée que par une étude classique de la relation pannes matérielles-manifestations fonctionnelles. L'approche la plus efficace, mais peut-être la plus difficile, consiste à partir de l'ensemble des pannes possibles et à associer à chaque panne sa manifestation fonctionnelle, qui peut être ou non détectée par les vérifications effectuées.

Exemple :

Dans l'exemple de la figure III - 9, on peut considérer que la détection assurée par le watch-dog implémenté dans le processus P3, vérifiant le temps entre deux passages dans la phase 1 et par le compteur de phases caractéristiques γ détecte 90 % des pannes de la partie contrôle du processeur. Par contre le contrôle de vraisemblance de la valeur de D ne détecte que 60 % des pannes de la partie opérative. On peut donc évaluer le taux de détection des pannes à environ $\mu 1 = 0,75$

b) taux d'isolation :

A chaque phase de contamination on associe l'ensemble E des erreurs qui peuvent contaminer l'extérieur. Si, avant toute phase ϕi de contamination on implémente une phase fictive de vérification, le taux d'isolation μ_2 (i) de cette phase ϕi est égal au taux des erreurs de E détectées par cette vérification.

Le taux d'isolation du processus μ_2 est égal à $\frac{1}{N} \sum_{i=1}^{N} \mu_2(i)$ si le processus contient N phases de contamination.

Exemple :

Le taux d'isolation de cet exemple est de l'ordre de 30%.

c) taux de dégradation μ_3 :

La dégradation des performances est dûe au temps supplémentaire passé à l'exécution des vérifications. Si t_1 (i) est le temps normal d'exécution d'un écoulement E , t_2 (i) le temps d'exécution avec les vérifications, mais en l'absence de pannes, le taux de dégradation associé à l'écoulement i est μ_3 (i),

$$\mu_3 (i) = \frac{t_2 (i) - t_1 (i)}{t_1 (i)}$$

La dégradation de performance pour le processus sera le taux moyen de dégradation des écoulements, pondéré par des facteurs de mix pi.

$\mu_3 = \Sigma$ pi . μ_3 (i)

si pi est la probabilité d'utiliser le $i^{ème}$ écoulement au cours de l'exécution du processus.

Exemple :

Si on affecte un temps unité à l'exécution d'une incrémentation, décrementation (γ:= γ + 1, E : = E - 1), ou chargement (E : = 32, γ : = 0) et à une procédure vide, un temps 2 à un échange d'information (B : = B_1, B_2 : = D...) et à la multiplication, un temps 4 au calcul d'une fonction complexe (D : = D + f' (B), D := f(B)), si la durée d'une transition est nulle, on obtient le tableau suivant pour le processus de la figure III 7, modifié pour obtenir le processus de la figure III 9 :

ECOULEMENT	E_1	E_2	E_3
t_1 (i)	7	10	12
t_2 (i)	8	11	14
μ_3 (i)	1/7	1/10	1/6
pi	0,2	0,4	0,4

$$\mu_3 = \sum_{i=1}^{3} pi\ \mu_3\ (i) = 0,14$$

III - 7 PRELOCALISATION MATERIELLE

La détection continue, bien que fonctionnelle, peut permettre une certaine localisation des pannes détectées. Cette prélocalisation peut s'effectuer à trois niveaux de finesse croissante.

a) 1er niveau

Quand une détection de bouclage est effectuée par un contrôle de type watch-dog (test de bouclage effectué par un autre processeur), le processeur observé comme le processeur observant peuvent être suspectés ainsi que le matériel de communication.

Le matériel de communication, généralement protégé par des codes ne sera réelement suspect que si plusieurs processeurs envoient une alarme. Le processeur observant utilisant peu de matériel et en général du matériel hardcore pour la vérification, on commencera par suspecter le processeur observé.

b) 2e niveau

Dans un processeur suspect, il est intéressant de localiser la panne dans la partie contrôle ou dans la partie opérative.

Un processeur est accusé par l'une des trois procédures de détection:

- test de passage illégal d'un écoulement à un autre : la panne est alors forcément dans la partie contrôle ; en effet, une panne de la partie opérative ne peut se traduire, au niveau du séquencement, que par un calcul de prédicat erroné ; ceci induit un mauvais cheminement, mais un cheminement possible dans le processus. Cette prélocalisation permet d'utiliser les méthodes de test de partie contrôle sous système par exécution d'écoulements standards, (18) .
- test de bouclage : cette erreur peut être provoquée par une panne de la partie contrôle (panne du matériel de séquencement) ou de la partie opérative (calcul erroné de prédicats), avec une probabilité beaucoup plus grande dans le premier cas. On commencera donc par un test de la partie contrôle.
- test des données : dans ce cas là, seule la partie opérative est impliquée dans un premier temps. Ici aussi, on pourra utiliser une méthode de test spécialisée pour la partie opérative, en utilisant le contrôle normal, ce qui peut se faire sous sytème très efficacement (19) .

c) 3e niveau

S'il est intéressant de localiser plus finement (faible intégration) on associe à un écoulement le matériel utilisé. Si ce matériel n'est pas le processeur complet, un test déterministe sera effectué sur ce sous-ensemble, si possible sans arrêt complet du système.

III - 8 CONCLUSION

A l'heure actuelle, le choix des moyens qui assurent une détection continue est en général intuitif et heuristique.

Nous avons proposé un modèle et une classification des phases d'un processus ainsi que des paramètres d'évaluation de l'efficacité des vérifications fonctionnelles ; cela devrait permettre un choix raisonné d'une politique de détection continue.

IV - CONTAMINATION ENTRE PROCESSUS

Dans ce paragraphe, nous étudierons la contamination des variables d'un processus par des valeurs erronées, reçues par le processus dans une phase de réception.

Un processus sera représenté par un réseau de Pétri, préstructuré, étiqueté, interprété, déterministe et déterminé.

IV - 1 CONTAMINATION A L'INTERIEUR D'UNE PHASE

A chaque phase i sont associés l'ensemble U^i des variables qu'elle utilise et l'ensemble D^i des variables qu'elle définit. U^i et D^i peuvent contenir des variables communes.

On définit l'ensemble Fi comme l'ensemble des variables erronées en sortie de la place i.

Un graphe de dépendance entre variables utilisées et variables définies est déterminé par analyse de la procédure associée à la phase ; un arc relie un élément de U à un élément de D si et seulement si la valeur de l'élément de D dépend de la valeur de l'élément de U. La relation de dépendance étant transitive, les chemins de longueur supérieure à 1 sont remplacés par une dépendance directe.

Etant donné l'ensemble des variables utilisées erronées U_f^i, les variables définies de façon erronée D_f^i en sortie d'une place i sont déterminées par le graphe de dépendance.

Exemple :

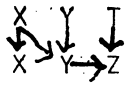
```
Procédure CALCUL (X,Y,Z,T).
Début
tant que X > 0 faire
début Y : = X+Y ;
      X : = X-1 :

fin ;
Z : = Y + T ;
fin CALCUL ;
```

U = {X,Y,T}

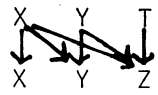
D = {X,Y,Z}

F : liste des variables erronées à l'instant initial est {Y,T,R}

Graphe de dépendance :



On supprime les chemins de longueur supérieure à 1 par compactage :



Donc, si la liste des variables erronées du processus était F = {Y,T,R} en entrée de la phase, elle devient F = {Y,Z,T,R} en sortie de la phase.

IV - 2 ETUDE DE DEUX PHASES EN SEQUENCE

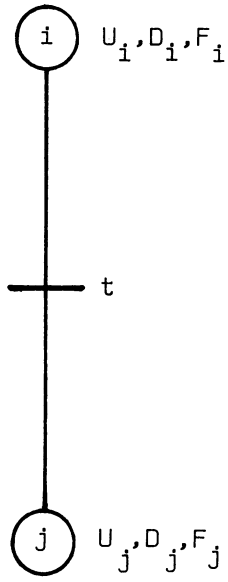


Figure III - 10

Deux cas se présentent :

- le prédicat t vaut toujours 1 : il suffit de propager la liste des variables erronées à travers la phase i pour obtenir la liste F_i, puis de propager F_i à travers la phase j pour obtenir F_j.
- le prédicat dépend d'une variable de synchronisation, qui est erronée. Si la valeur erronée est 0 au lieu de 1, le processus est bloqué ; si c'est 1 au lieu de 0, les procédures de toutes les places accessibles à partir de i seront exécutées à tort. Dans les deux cas, on peut considérer que toutes les variables définies en aval de i sont fausses. On inclura dans la liste des variables fausses toutes les listes D_k de variables définies en aval de i.

IV - 3 ETUDE DES NOEUDS ET

Définition :

Soit une transition t réalisant une distribution; on définit la transition t^* comme la transition réalisant la jonction associée. (Fig.III-11).

t^* est la transition

α) distinctedet

β) qui appartient à tous les chemins de t vers I

γ) telle que toute autre transition ayant les propriétés α,β appartienne à un chemin de t^* vers I.

t^* est donc unique.

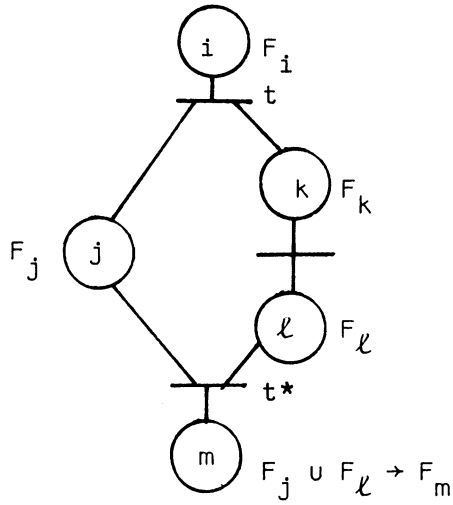


Figure III.11

Existence de t^*

Le graphe étant préstructuré, la transition to (transition de retour) vérifie les propriétés α et β. Donc t^* existe : c'est à la limite la transition to.

$$\begin{cases} t_3 = t_1^* \\ t_3 = t_2^* \end{cases}$$

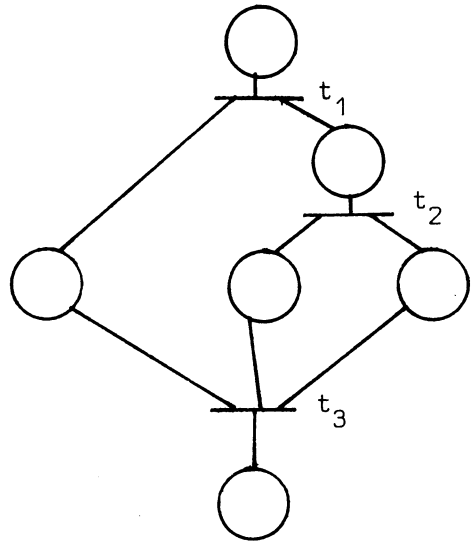


Figure III.12

Une même transition peut être la transition "étoile" de plusieurs transitions (Fig. III-12).

Algorithme de contamination (voir Figure III.11)

La liste F_i est propagée indépendamment dans chacune des branches ({j} et {k, t_1,ℓ}), jusqu'à t*. L'union des listes Fj et Fℓ obtenues dans chaque branche est alors réalisée à l'entrée de m. Le réseau étant déterminé, la propagation peut se faire indépendamment sur les branches (les variables définies dans une branche ne peuvent être utilisées dans une autre).

IV - 4 ETUDE DES NOEUDS OU

Définition

Soit une place réalisant une sélection; on définit la place p^* comme la place réalisant l'attribution associée (Fig.III.13).

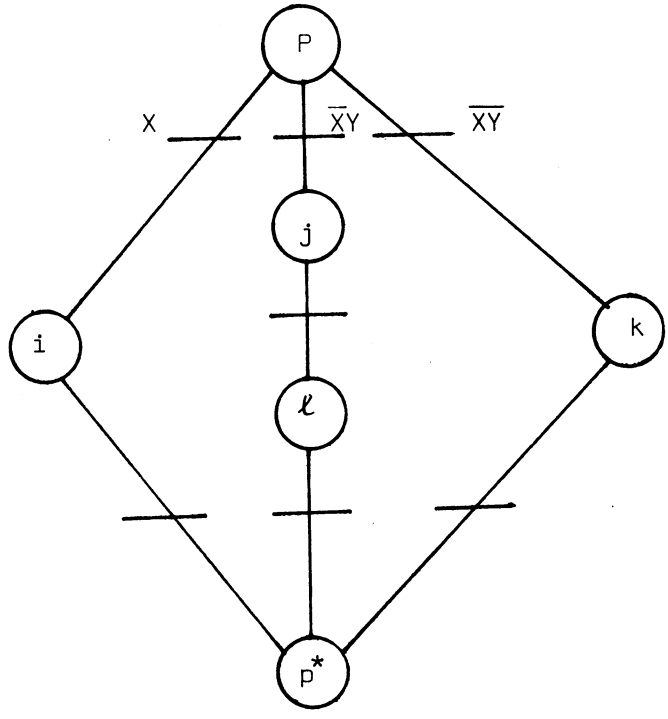


Figure III.13

p* est la place :

δ distincte de p

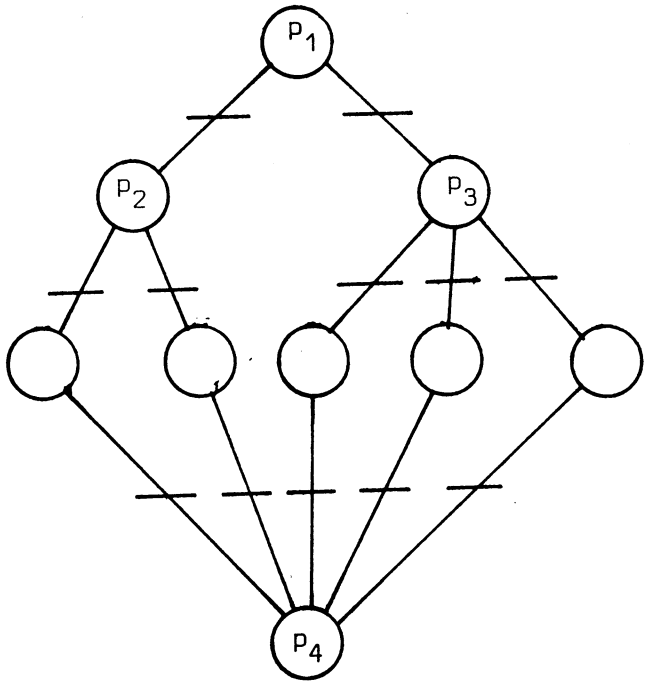
ε qui appartient à tous les chemins de p à F

ζ telle que toute autre place vérifiant δ et ε appartienne à un chemin de p* à F.

p* est donc unique.

Existence de p*

Le graphe étant préstructuré, la place F vérifie les propriétés δ et ε. Donc p* existe : c'est à la limite F.



$$\begin{cases} p_4 = p_1^* \\ p_4 = p_2^* \\ p_4 = p_3^* \end{cases}$$

Figure III.14

Une même place peut être la place "étoile" de plusieurs places réalisant une sélection (Fig. III-14).

Algorithme de contamination

Nous considérons deux cas, suivant que les prédicats dépendent d'une ou plusieurs variables erronées ou non. A chaque prédicat est associée la liste V des variables dont il dépend; on distingue deux cas :

$V \wedge F = \phi$

F étant la liste des variables erronées; dans ce cas le prédicat est correct. On examine successivement la contamination dans les diverses branches du OU, en considérant les branches comme indépendantes.

$V \wedge F \neq \phi$

Les prédicats pouvant être erronés, une branche peut être exécutée à tort; même si les variables définies dans cette branche ne dépendent pas directement des variables de F, elles peuvent toutes être erronées car calculées à un instant non prévu. On considèrera donc comme erronées toutes les variables définies dans les branches initialisées par un prédicat erroné, ceci jusqu'au passage à travers p*.

Exemple (voir figure III.13)

$Y \in Fp$. Donc les variables définies dans les deux branches (j,ℓ) et (k) sont erronées. En entrée p* on réalise l'union des listes Fi, Dj, $D\ell$, Dk, Fp, et on propage cette liste à travers p* (voir § IV.2).

IV - 5 ETUDE DE LA CONTAMINATION

Les algorithmes de contamination peuvent être utilisés de diverses façons.

IV - 5a ETUDE DE LA SENSIBILITE D'UN PROCESSUS A UNE ERREUR D'ENTREE

On peut étudier l'étendue de la contamination. Dans certains cas, le processus n'est que partiellement perturbé par une erreur en entrée : le processus joue le rôle de buffer par rapport à cette variable. Il est évident que, dans ce cas, la perturbation se produit, mais dans un autre processus.

Exemple

Si nous reprenons le processus de la figure III.7, en l'interprétant à l'aide des graphes de dépendance, nous obtenons le graphe de la figure III.15.

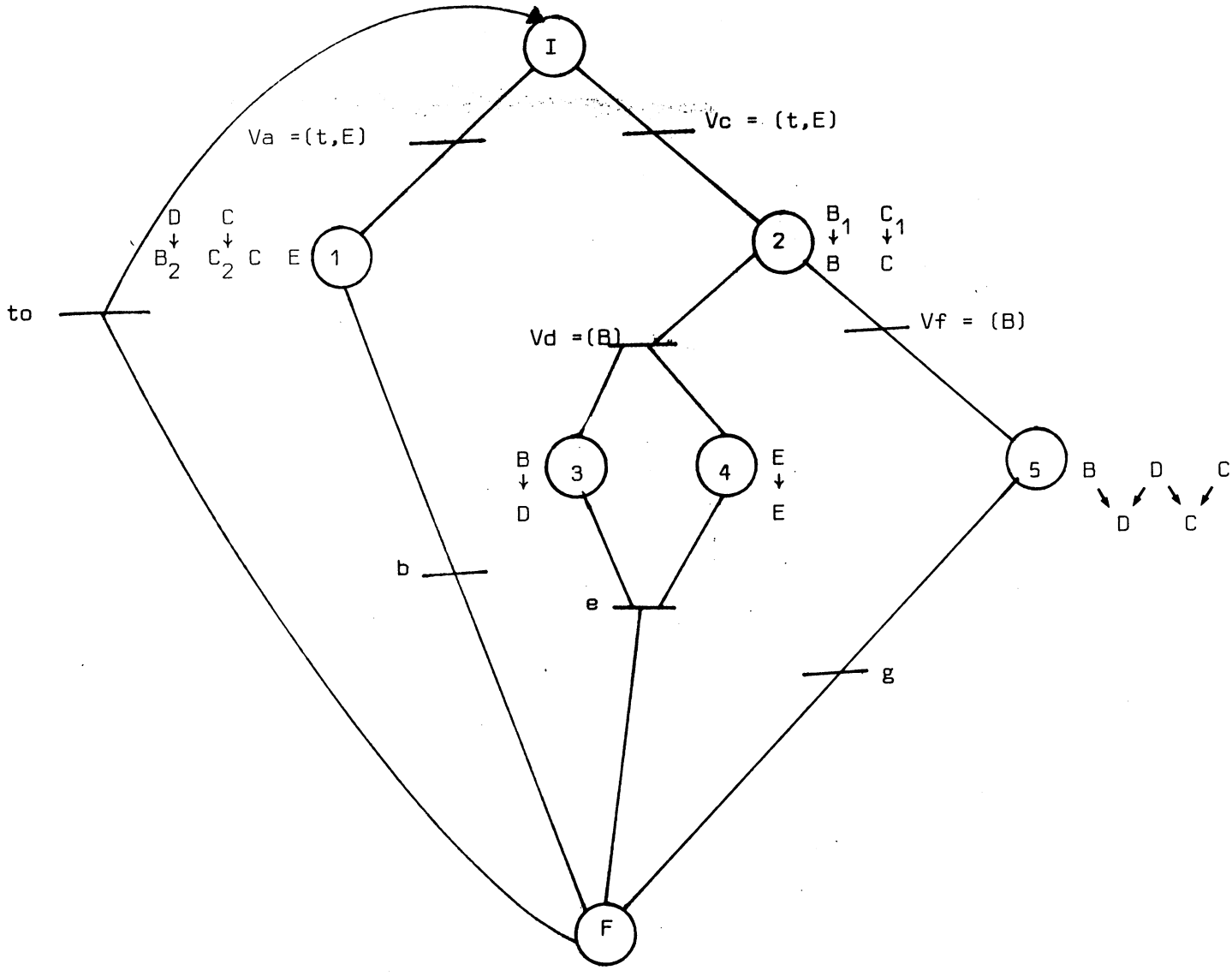


Figure III.15

- Sensibilité du processus à une valeur erronée de B_1.

  La place 2 réalise une sélection et $(2)^*$ = F.

  Donc, les valeurs définies en 3, 4, 5 sont erronées.

  Au premier passage dans la place F, la liste des variables erronées est

  $\{B_1, B, D, E, C\}$

  La place I réalise une sélection. Les prédicats dépendent de E qui est erronée. Donc, au 2e passage dans la place F, la liste des variables erronées est $\{B_1, B, B_2, C, C_2, D, E\}$.

Si T est le temps maximum d'exécution d'un écoulement quelconque, le processus est entièrement contaminé au bout de 2T.

- Sensibilité du processus à une valeur erronée de C_1.
  Au premier passage dans la place F, seules les variables C et C_1 sont erronées.
  Au 2e passage dans F, C, C_1 et C_2 sont erronées, et c'est le maximum.
  La sensibilité du processus à une erreur sur C_1 est faible.

IV - 5b ETUDE DE LA LATENCE D'ERREUR

Une variable est quelquefois utilisée longtemps après avoir été acquise par le processus. La détection par des moyens logiciels se produira donc tardivement, ce qui rend la reprise particulièrement complexe. Par l'étude de la contamination, la latence d'erreur peut être estimée de façon précise et une politique de choix des points de reprise peut être déterminée.

IV - 5c ETUDE DE LA DECONTAMINATION

Une valeur fausse d'une variable d'entrée contamine le processus; après acquisition d'une nouvelle valeur de cette variable ayant alors une valeur juste, le processus peut être "décontaminé", c'est-à-dire que la nouvelle valeur se propageant dans le processus, les variables contaminées peuvent acquérir une valeur juste. Si le système peut tolérer des valeurs temporairement fausses, ou bien la perte de quelques valeurs, cette étude permet de déterminer si la décontamination se produit dans des limites de temps acceptables par le système.

Exemple

Si on considère le processus de la figure III.15, on obtient :

- la décontamination après une valeur juste de C_1 est effectuée au bout de 100 T environ.

On a C, C_1 et C_2 faux. C devient juste dès l'acquisition de C_1 juste. Il faut passer dans la place 1 pour décontaminer C_2. Les valeurs positives et négatives ou nulles de B étant équiprobables et E étant initialisée à 32, on passe en moyenne tous les 64 T à la place 1. Donc, au bout de 100 T, le processus est presque certainement décontaminé.

- la décontamination après une valeur fausse de B_1 suivie d'une valeur juste ne peut être effectuée.

En effet, la variable E mémorise les séquencements effectués dans le processus et donc tiendra compte des séquencements effectués quand la valeur de B était erronée. Il faudra donc dans tous les cas effectuer une reprise (qui peut être simplement une réinitialisation du processus).

V - CONCLUSION

Dans ce chapitre, nous avons développé un modèle qui permet, d'une part le choix d'une politique de détection continue, d'autre part l'étude de la contamination d'un processus. Cette étude sera étendue pour permettre la détermination d'une politique de reprise et, si nécessaire, le choix de points de reprise efficaces.

CHAPITRE IV

DETECTION DISCONTINUE

I - INTRODUCTION

La détection discontinue consiste à faire exécuter à un processeur des programmes de test, soit durant une période d'inactivité, soit en interrompant temporairement l'activité du processeur. Cette détection doit rester transparente fonctionnellement.

Pour que le processeur en état de test continue à remplir ses fonctions pour le compte du système global, on peut envisager deux solutions :

- soit le temps consacré au test est très court. Ceci est obtenu grâce à une écriture très modulaire du programme de test (concaténation de séquences de test) et à des possibilités d'interruption de ce programme en fin de séquence ou des séquences elles-mêmes. Cette réalisation modulaire du programme de test est rendue possible, par une méthodologie d'écriture de programme de test de type structurel définie en (21).

- soit le processus peut être exécuté par un processeur différent. Durant le temps de test, toute tâche affectée à ce processeur est automatiquement déroutée vers un autre processeur. Cela est possible quand le système est modulaire.

En II l'interaction du programme de test et du processus exécuté par un processeurs est modélisé. En III une stratégie de détection discontinue est définie pour un système modulaire : le multimicroprocesseur CII.

II - DETECTION DISCONTINUE

Le processus est modélisé par un réseau de Pétri pré-structuré.

II - 1 CHOIX DU PROGRAMME DE TEST

a) test par écoulement standard

L'exécution du programme de test consiste alors à exécuter un ou plusieurs écoulement normaux du processus avec des opérandes spécifiques qui sont en fait des données de test élaborées de façon préalable sur le mode déterministe (21). Le programme de test respecte donc le séquencement normal du processus, ce qui peut rendre impossible un test exhaustif.

Il s'agit d'une approche de test fonctionnelle descendante. L'exécution peut être initialisée (choix de l'écoulement, stockage des données de test) soit par un processeur externe spécialisé qui recueille et analyse les résultats, soit par le processeur lui même, qui comprend alors un "hard-core" d'initialisation et de comparaison

Exemple :

Dans le centre de commutation téléphonique E_{10} (22).

Cela consiste à envoyer une ou plusieurs communications fictives imbriquées dans les communications normales à partir d'un organe spécialisé ou modifié pour la maintenance.

b) test par écoulement spécialisé

Dans ce cas, un nouvel écoulement spécialisé du processus est créé dès la conception. Cet écoulement est élaboré à partir de techniques d'écriture de programme de maintenance très efficaces (approche structurelle et déterministe ascendante (21) et non plus fonctionnelle descendante comme dans le cas précédent). Un test exhaustif accompagné d'un diagnostic fin peut être obtenu facilement.

Dans les deux cas, le test doit respecter la contrainte de transparence.

II - 2 STRATEGIES DE TEST

Les différentes stratégies de test sont représentées sous la forme d'un arbre de choix.

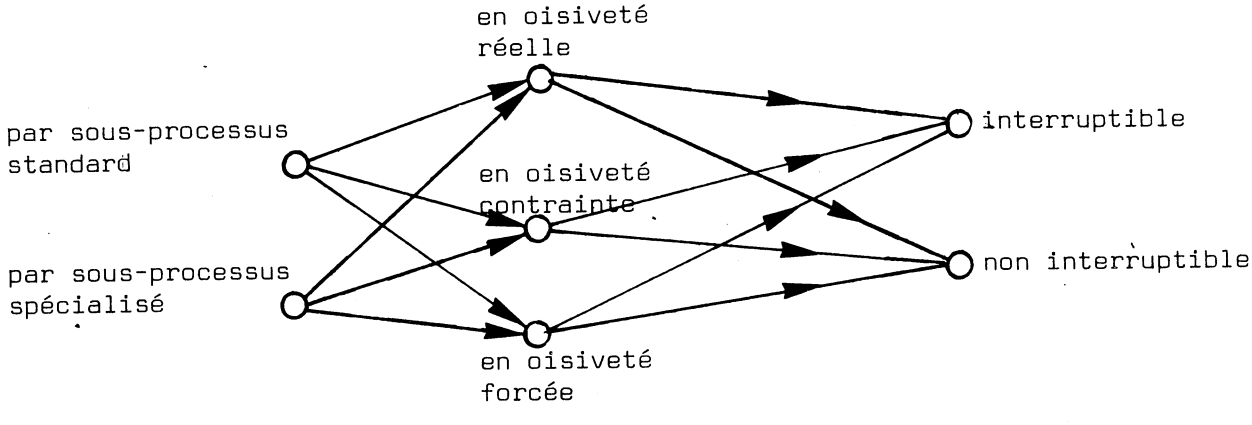


Figure IV - 4

Considérons la phase initiale I du processus et les prédicats initiaux (t1,...,tn) qui sélectionnent un des écoulements standards.

a) test en oisiveté réelle non interruptible

Si aucun des prédicats (t1,...,tn) n'est vérifié, le processeur est oisif. On initialise un programme de test.

Si le test est exécuté par un écoulement spécialisé, cet écoulement est exécuté puis le processus revient à la phase finale F. (Fig. IV - 2)

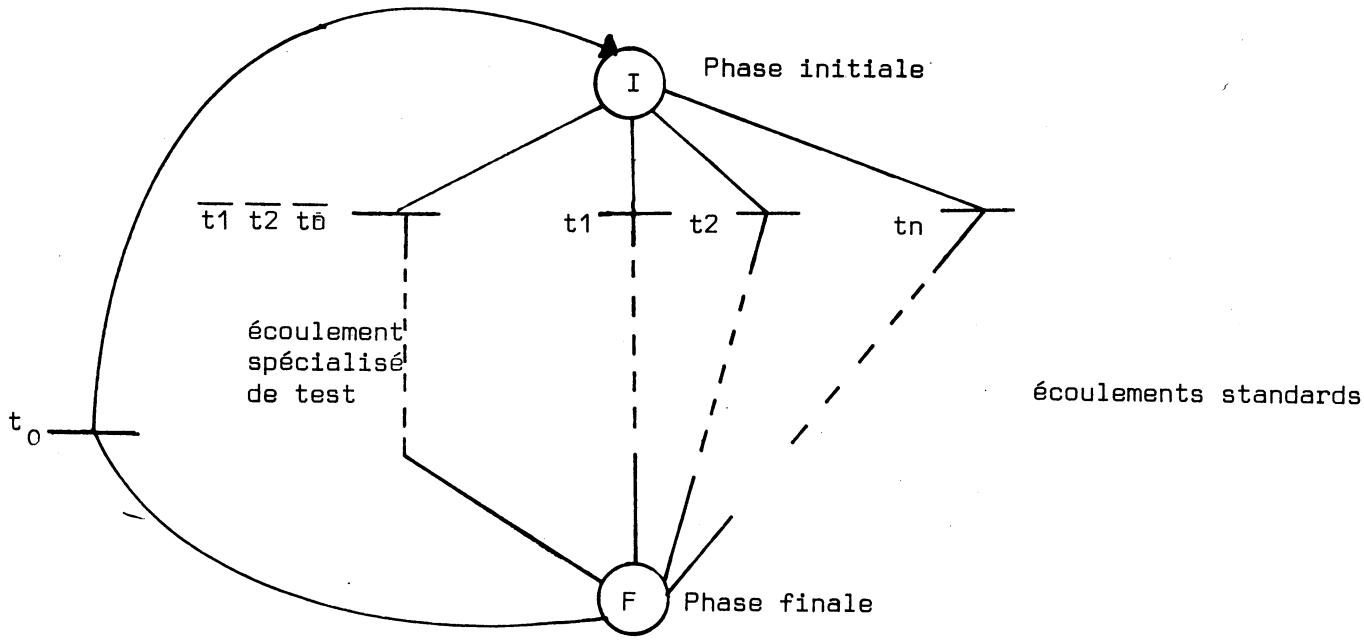


Fig. IV - 2

Si le test est exécuté par un écoulement standard commandé par le processus, le processus doit initialiser le test, exécuter l'écoulement standard, puis analyser les résultats. Pour cela, il faut disposer d'une bascule de test T qui sera positionnée à 1 durant l'initialisation du test. Dans la figure 2, on suppose que l'écoulement standard à exécuter pour le test est l'écoulement initialisé par le prédicat t_1.

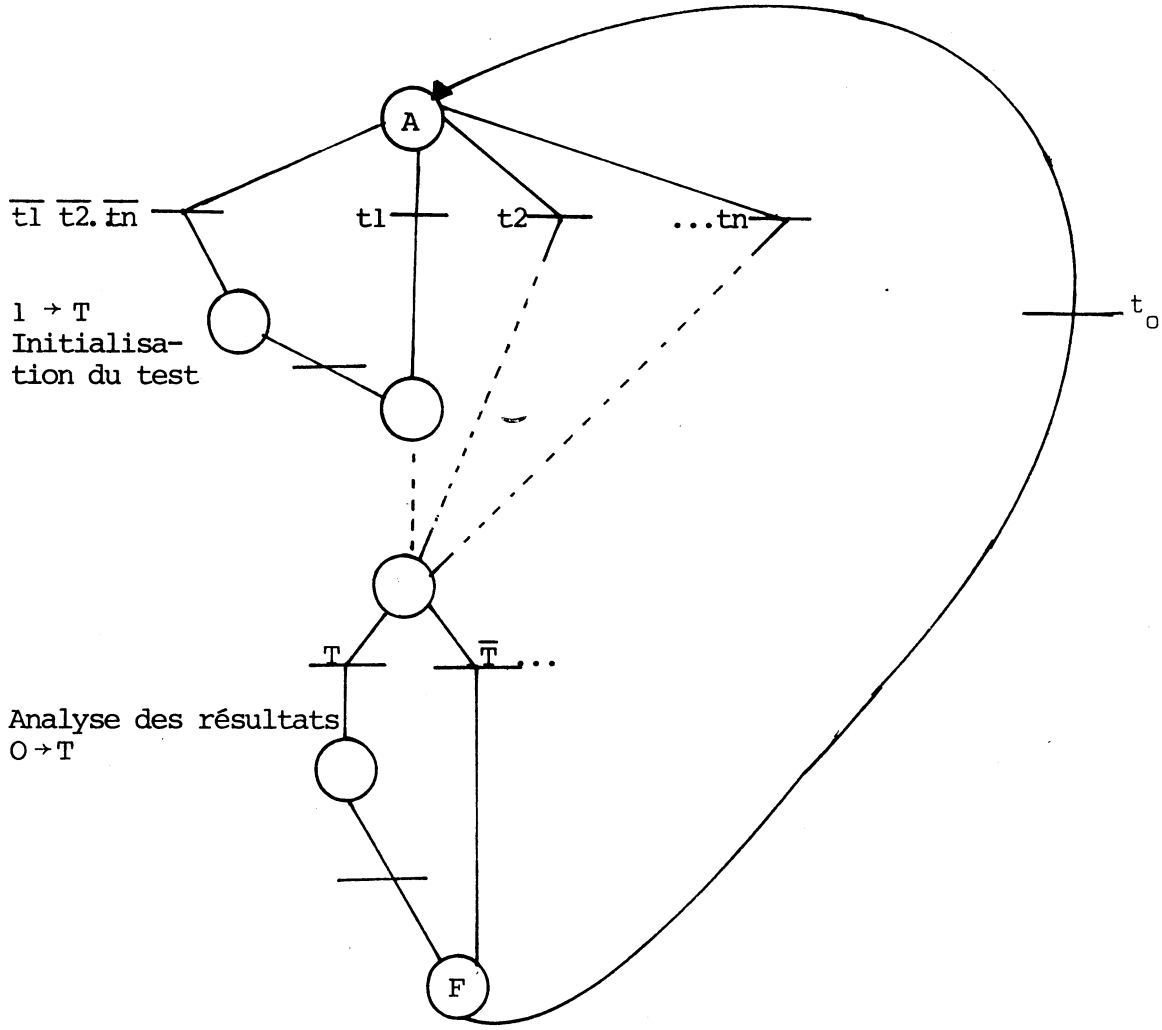


Fig. IV - 3

b) test en oisiveté réelle interruptible

Le principe est le même, mais après chaque phase de test la transition à la phase suivante de test n'est mise à feu que si le système est toujours oisif.

Dans le cas d'un écoulement spécialisé, chaque phase est du type :

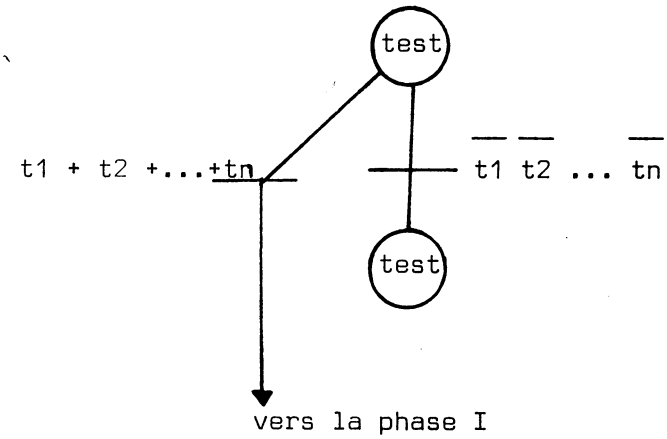


Fig. IV - 4

Dans le cas d'un écoulement standard, on obtient, si tk est le prédicat de passage de la phase i à la phase j en fonctionnement normal,

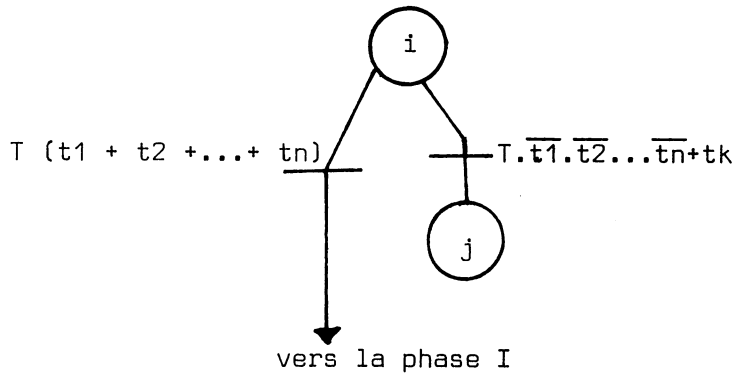


Fig. IV - 5

c) test en oisiveté contrainte

On impose au processeur l'exécution d'un programme de test quand un prédicat R est vrai.

Le prédicat porte en général sur le temps écoulé depuis le dernier test. Ce test respecte la fonction du processeur, c'est-à-dire n'interrompt pas l'exécution d'un écoulement, mais est exécuté entre deux écoulements.

On obtient :

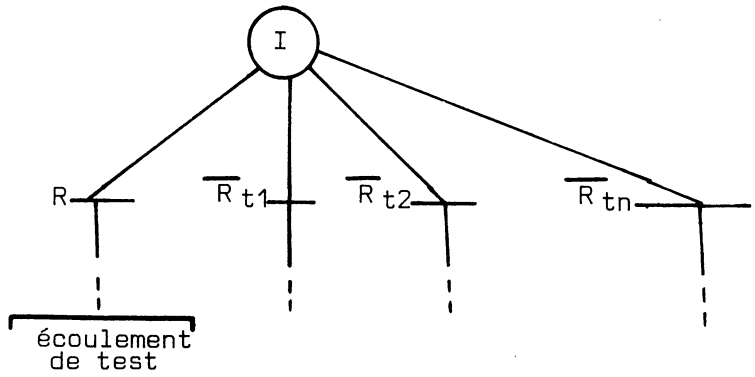


Fig. IV - 6

d) test en oisiveté forcée

On peut imaginer de forcer le processeur à exécuter un programme de test dès que un certain prédicat F est vrai. Ce prédicat peut être positionné par le monde extérieur sur détection d'une erreur. Ce test interrompt donc l'exécution des écoulements.

Chaque phase du processus sera donc du type :

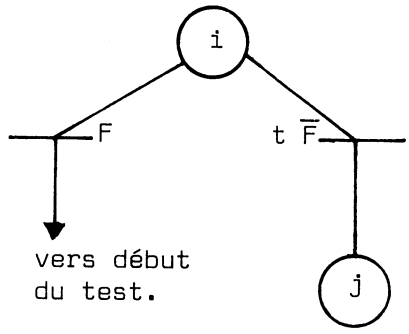


Fig. IV - 7

e) test par écoulement standard commandé par un organe externe

Ce test est totalement transparent pour le processeur testé.

II - 3 EFFICACITE

L'efficacité d'un test discontinu en cours de fonctionnement peut être

mesurée par 4 paramètres : le taux de pannes détectées μ_1, le taux d'isolation μ_2 le taux de dégradation des performances μ_3, et le temps maximum D au bout duquel une panne détectable est à coup sûr détectée (latence de détection).

a) le taux de pannes détectées μ_1, dépend entièrement du type de test réalisé. Il sera important ($\geq$ 90 %) dans le cas d'une approche structurelle ascendante, c'est à dire dans le cas où un écoulement spécialisé pour le test est créé ; dans le cas d'une approche fonctionnelle descendante (écoulement standard) il sera plus faible.

b) taux d'isolation μ_2

Le taux d'isolation est nul ou à peu près nul. On ne peut assurer qu'une erreur est détectée avant qu'elle contamine un autre processus.

c) latence de détection D

La latence dépend de la fréquence d'exécution du test. Pour un test exécuté en oisiveté réelle, cette fréquence dépend de la charge du système; dans le cas d'oisiveté contrainte, elle dépend du prédicat R.

d) taux de dégradation μ_3

Ce taux dépend de la longueur et de la fréquence d'exécution du programme de test. Pour un taux donné de pannes détectées, un programme de test structurel sera plus court qu'un programme de test fonctionnel.

La latence de détection D et le taux de dégradation μ_3, dépendant de la charge du système, ne peuvent être évalués que par des mesures soit de simulation, soit prises sur le système réel.

III APPLICATION : SYSTEME MULTIMICROPROCESSEUR CII

Nous commençons par donner une présentation succinte du système. Seules les particularités qui nous intéressent sont présentées ; les détails peuvent être trouvés dans le rapport du contrat DRME n° 73/848.

Nous étudions ensuite une politique de détection discontinue, bien adaptée à la structure du système ; nous verrons que si certaines des caractéristiques sont gênantes d'un côté, elles sont intéressantes de l'autre : par exemple, l'impossibilité d'adresser et d'interrompre les processeurs, qui rendent la fréquen-

ce de détection plus aléatoire mais permettent une reconfiguration facile du système logiciel.

III - 1 PRESENTATION SUCCINCTE DU SYSTEME MULTIMICROPROCESSEUR CII

a) structure matérielle

Ce système est formé d'un certain nombre de processeurs logiques qui regroupent les capacités de traitement de même type. Chaque processeur logique est donc un ensemble de processeurs physiques identiques.

Ce système est conçu pour le traitement en parallèle ; en fait, il existe deux niveaux de parallélisme :

- entre processus,
- entre miniprocessus.

Ce système traite les deux niveaux. Un processeur logique est formé de plusieurs molécules, qui exécutent des processus en parallèle ; une molécule est formée de plusieurs atomes qui exécutent des miniprocessus en parallèle.

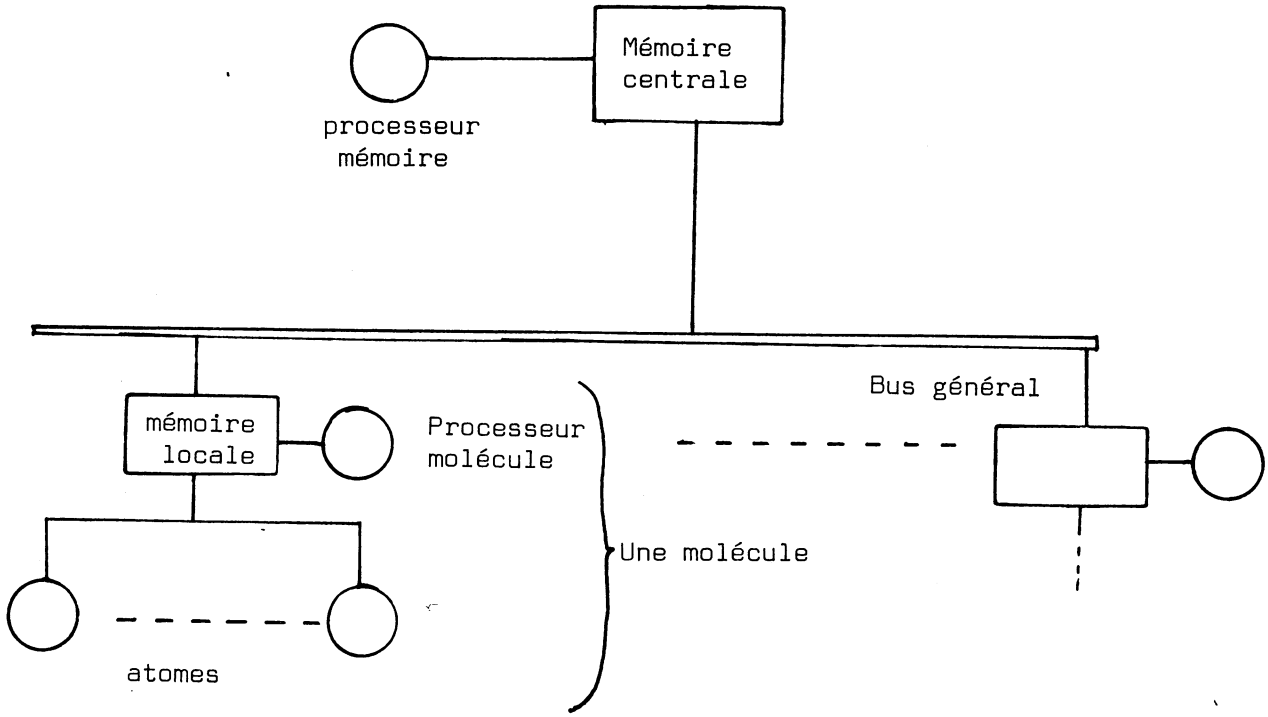


Fig. IV - 8

Une molécule est formée :

- de plusieurs atomes qui sont des microprocesseurs,
- de la mémoire locale,
- du processeur molécule qui gère les échanges
- du bus local.

La mémoire locale comprend un cache associatif et des zones de mémoire pour la gestion de la molécule.

b) allocation des processus aux processeurs

A un processeur logique est associée une file d'attente. La création et la destruction d'un processus sont faites par les primitives FORK et JOIN.

Un processus nouvellement créé est placé dans la file d'attente du processeur logique sur lequel il doit s'exécuter. A cet instant on envoie un signal ("sonnette") à tous les processeurs physiques. Ceux d'entre eux qui sont inactifs consultent la file d'attente de leur processeur logique et le 1er acquiert le 1er processus de la file. Puis si elle est vide, il baisse le niveau de la sonnette.

Symétriquement, il existe une "anti-sonnette" ou signal halt ; elle est testée par les processeurs actifs. Si le "halt" est levé, le processeur va comparer le nom du processus qu'il est en train d'exécuter avec le ou les processus à arrêter. En cas d'identité, le processeur abandonne son processus et devient inactif.

Un processus n'est actif que sur une molécule à la fois.

c) traitement des mini-processus

Un processus est éclaté en plusieurs mini-processus parallèles par un minifork. Les mini-processus sont exécutés par les processeurs d'une même molécule.

La gestion des miniprocessus se fait de façon légère. La structure d'arbre générée par les miniforks est mémorisée par des compteurs et des pointeurs au niveau de la mémoire locale. Le processus redevient séquentiel par un minijoin

quand toutes les branches de l'arbre sont exécutées. Il existe une minisonnette et une mini antisonnette au niveau de la molécule.

d) particularités de ce système

Un processeur atome n'est pas adressable : il n'est identifié que par le nom du processus qu'il exécute. Tous les processeurs sont équivalents.

L'allocation des processus aux processeurs n'est ni autoritaire ni centralisée. C'est le processeur qui décide de prendre en charge un processus, sachant qu'il y a des processus en attente dans la file.

Le système ignore le nombre de processeurs dont il dispose. Les processeurs ne sont interruptibles que par l'antisonnette ou l'antiminisonnette. Cette interruption se fait sans sauvegarde du contexte. En cas de défaut de page ou autre, le processeur reste en attente de la résolution de la faute de page.

III - 2 PRINCIPE DE LA DETECTION

Nous nous intéressons ici à une structure non redondante. Le but est d'assurer une certaine sécurité à un coût minimum pour des applications non critiques. Cela sera obtenu avec une politique de test adaptée à la structure du système, associée à une politique de remise en état des données et de reprise des programmes après détection d'une panne. Plus la détection sera rapide, moins le redémarrage sera complexe.

Il faut donc :

- détecter les pannes,
- localiser les pannes,
- reconfigurer le système matériel,
- reconfigurer le système logiciel.

La détection des pannes sera étudiée en détail ici.

La localisation des pannes se fera au niveau d'un processeur atome ou d'une molécule. Une localisation plus fine en cours de fonctionnement est inutile : le processeur atome ou la molécule sont non reconfigurables. Une localisation plus fine sera faite au cours d'une maintenance normale.

La reconfiguration du système matériel ne pose pas de problème. Après détection d'une panne dans un élément (atome ou molécule), la suppression de cet élément se fera par autoblocage (commandé par le hard-core) de l'interface de l'élément avec le reste du système.

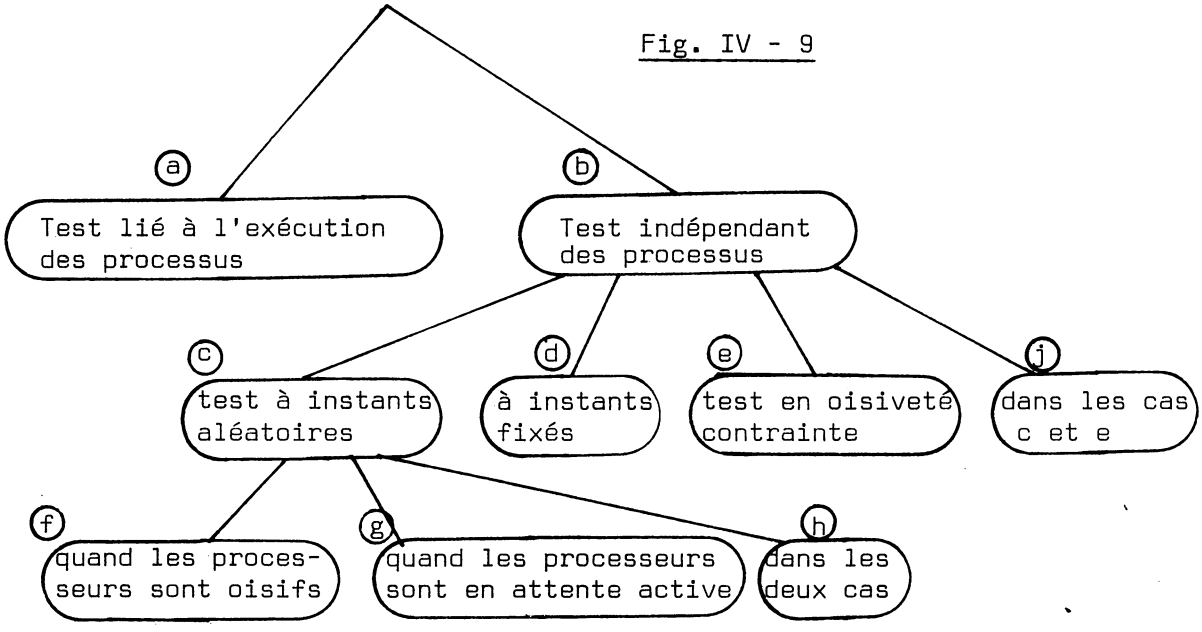
La reconfiguration du logiciel est rendue inexistante par la structure du système. Le système ignorant le nombre de processeurs dont il dispose et ne pouvant les adresser, la déconnexion d'un élément est transparente. La structure originale du système est très intéressante dans ce cas.

IV - CHOIX POSSIBLES POUR LE TEST DES PROCESSEURS ATOME

Cette politique de test doit permettre de tester fréquemment les processeurs tout en diminuant le moins possible les performances du système.

Différents compromis sont possibles avec des caractéristiques différentes. Après simulation, les solutions intéressantes seront retenues. Les choix possibles sont présentés sous la forme d'un arbre. Chaque noeud est explicité, les avantages et inconvénients de chaque choix sont donnés, ainsi qu'un algorithme de réalisation. La réalisation matérielle se fait pour la plus grosse part au niveau du séquenceur, dont la complexité est augmentée.

Fig. IV - 9



Arbre de choix

IV - 1 TEST LIE A L'EXECUTION DES PROCESSUS

La solution (a) consiste à tester tout le matériel utilisé par un processus avant de le valider et d'en donner les résultats.

La sécurité obtenue est très grande : protection contre toutes les pannes permanentes. Cette solution est-elle réalisable ?

Un processus utilise :

- processeur d'E/S et périphériques,
- au moins un processeur atome,
- les autres processeurs de la même molécule en cas de mini fork,
- d'autres molécules en général,
- la mémoire centrale.

Pour garantir une bonne sécurité, il faut qu'à chaque "fin de séquence" (définie en (23)) tout le matériel utilisé depuis le dernier test soit testé et trouvé en état de marche.

Cela veut dire que

- une molécule n'exécute qu'un processus au plus (pour pouvoir tester les atomes ayant exécuté les branches d'un mini fork),
- que le système passe un temps appréciable en état de test, les séquences pouvant être très courtes par rapport à la longueur de test.

Cette solution est caractérisée par :

- avantages : protection contre toutes pannes permanentes,
- inconvénients : dégradation importante des performances du système.

Une solution redondante paraît préférable :

- les pannes aléatoires comme permanentes sont détectées,
- la dégradation des performances est sans doute à peu près équivalente sinon moindre (ceci en comparant deux systèmes ayant le même nombre de processeurs en tout).

Le choix est donc à écarter. Avec une sécurité meilleure, une solution redondante semble plus faisable et moins coûteuse.

IV - 2 TEST INDEPENDANT DES PROCESSUS

La solution (a) étant écartée, nous choisissons de faire exécuter un programme de test aux processeurs de façon indépendante des processus.

Dans ce cas, on ne peut pas garantir le non-propagation des erreurs : en particulier des données globales ou des tables systèmes peuvent être modifiées à tort par un processeur en panne.

La plus mauvaise solution garantit la déconnection des processeurs en panne quand le système est oisif (c'est à dire en le laissant "tourner à vide" un certain temps). Cela permet, en cas de résultats aberrants, de reconfigurer le système en déconnectant les unités en pannes sans intervention d'une équipe de maintenance spécialisée. La disponibilité globale du système est ainsi augmentée. Un certain nombre de travaux doivent être réexécutés après la reconfiguration du système. La sécurité de cette solution est faible.

Cette plus mauvaise solution est le noeud (f) que nous explicitons plus loin.

IV - 3 TEST A INSTANTS ALEATOIRES

Le test se faisant à des instants liés à la charge du système, cette solution respecte les performances du système.
Par contre, aucune garantie ne peut être donnée quand à la fréquence du test.

Le test peut être exécuté :

- quand les processeurs sont oisifs (noeud (f)).
- quand les processeurs sont en attente active (noeud (g)).
- dans les deux cas (h). En fait on peut choisir la solution (f) mais le choix de (g) implique le choix de (f)

IV - 4 TEST EN OISIVETE REELLE

a) principe

Dans ce système multiprocesseur, on peut s'attendre à ce que les processeurs ne soient pas tous actifs à un même moment, étant donné qu'il est diffi-

cile d'exhiber un degré de parallélisme suffisant.

Nous proposons donc que les processeurs oisifs exécutent un programme de test. Il suffit que ce programme détecte les pannes. La localisation de ces pannes n'est pas nécessaire puisque la détection d'une panne conduira à l'inhibition ("suicide") d'un processeur en cause, L'équipe de maintenance aura la charge de détecter plus précisément et de remplacer l'élément du processeur qui est en panne.

Le programme de test, sans localisation, sera exécuté en un temps relativement bref : bref par rapport au temps d'exécution d'un processus, mais sans doute long par rapport à la durée d'exécution d'une branche d'un mini forK.

Pour ne pas diminuer le degré de parallélisme fin, nous proposons que le programme de test soit interrompu à chaque appel des sonnettes locales ou générales.

b) interruption du programme de test

A tout appel de la sonnette (locale ou générale) le test est interrompu.

On pourrait sauvegarder l'adresse de l'instruction de test suivante. Le programme de test serait alors le programme "normal", toute exécution d'un processus serait considérée comme traitement d'une interruption. La zone de sauvegarde des adresses de retour serait plus grande et la sauvegarde serait une perte de temps.

Le test sera donc interrompu sans sauvegarde. Cela peut se faire simplement par action sur le séquenceur.

Il faut créer un état "test" qui sera mémorisé dans une bascule test.

- test = 0 , état d'exécution,
- test = 1 , état test

interruption du test

Si test ∧ sonnette haute alors

test ← 0

allera prise en compte de la sonnette

Sinon continuer le programme de test

mise en état de test

Si fin du mini processus ∧ sonnette basse alors

test ← 1
allera initialisation du test
sinon si sonnette haute allera prise en compte de la sonnette.

Ceci se fait en une microinstruction.

Cette méthode permet donc que les processeurs oisifs soient testés, ceci sans perte de performance pour le système.

c) programme de test divisé en séquences

Il faut garantir que le programme de test soit exécuté dans sa totalité à une certaine fréquence : on ne peut pas permettre que seul le début du ppogramme soit exécuté, en cas d'interruptions fréquentes.

Ce programme doit être divisé en séquences assez courtes. Après interruption le programme sera repris à la première séquence non encore exécutée dans sa totalité, en supposant que les éléments testés dans les séquences précédentes ne soient pas tombés en panne entre temps, ce qui est une hypothèse raisonnable.

Ces séquences n'ont entre elles comme relation que leur ordonnancement. Il faut donc :

1) Etudier le programme de test en tenant compte de la nécessité du découpage en séquences,
2) Réaliser le mécanisme qui permettra de savoir quelle séquence est à effectuer.

Le point (1) ne pose pas de problème . Le découpage en séquences du programme sera le reflet du découpage du processeur en microblocs qui est effectué au cours de l'étude du programme de test (21).

Le point (2) est résolu par la création d'un registre dont le contenu indique la séquence qui doit être exécutée. A la fin de chaque séquence, le contenu du registre sera modifié pour pointer sur la séquence suivante.

Le registre peut :

- indiquer le numéro d'ordre de la séquence à exécuter,
- avoir le n^e bit à 1 si la n^e séquence est à exécuter,
- pointer vers l'adresse de début de la séquence à exécuter.

La dernière solution étant plus rapide, c'est celle que nous avons choisi. Ce registre sera appelé par la suite registre de test.

L'exécution du programme de test par les processeurs oisifs :

- ne change rien aux performances du système,
- mais la fréquence du test dépend de la charge du système.

En cas de grande activité, cette fréquence devient très faible. La solution f garantit que le système étant oisif, les processeurs en panne seront déconnectés, sans manipulations complexes. Le système est donc reconfiguré automatiquement, mais des erreurs peuvent être propagées.

IV - 5 EXECUTION D'UN PROGRAMME DE TEST PAR LES PROCESSEURS EN ATTENTE ACTIVE

Dans ce système, en cas de faute de page, les processeurs restent en attente de la résolution de cette faute de page. Cette attente s'appelle attente active; le contexte est conservé. Si les processeurs sont avertis du fait qu'ils sont en attente active, ceci pour qu'ils ne surchargent pas le bus général par leurs demandes, nous pouvons utiliser l'attente active pour leur faire exécuter un programme de test.

Ce programme de test sera évidemment interruptible, pour que les processeurs puissent reprendre leur activité normale, suivant la même méthode que plus haut.

L'état test en attente active est différent de l'état "test en oisiveté", défini plus haut.

En effet, avant l'exécution du programme test, il faut sauvegarder le contexte; à l'interruption du test, il faut le restaurer.

Les algorithmes sont différents :

Mise en état "test en attente active"

Si attente active alors
sauvegarder le contexte
état = test en attente active
allera début du test

interruption du test

Si test en attente active ∧ arrivée de la page alors
arrêter le test
restaurer le contexte
allera exécution normale

Si un processeur est en panne et est déconnecté, le"watch-dog" général relancera le processus sur un autre processeur.

La solution (g) :

- implique qu'il faut prévoir la sauvegarde et la restauration du contexte,
- dégrade légèrement les performances du système, puisqu'il faut restaurer le contexte après l'interruption de test,
- garantit une fréquence importante de test en cas de charge importante du système. En effet, dans ce cas là, les fautes de pagès deviendront plus fréquentes.

IV - 6 EXECUTION D'UN PROGRAMME DE TEST PAR UN PROCESSEUR EN ATTENTE ACTIVE OU EN OISIVETE REELLE

La solution (f) garantit une bonne fréquence de test dans les cas où le système est peu chargé ; la solution (g) dans le cas où le système est très chargé ; on peut espérer obtenir une fréquence de test à peu près constante en exécutant des programmes de test quand les processeurs sont oisifs et quand ils sont en attente active.

Un processeur aura 5 états possibles :

- test en oisiveté,

- test en attente active,
- oisiveté,
- activité
- attente active.

Si oisiveté ∧ sonnette basse alors
état = test oisiveté
allera contenu du registre de test
Sinon si sonnette haute alors
allera prise en compte sonnette.

Si attente active alors
sauvegarder le contexte
état = test en attente active
allera contenu du registre de test

Si test en oisiveté ∧ sonnette haute alors
état = actif
allera prise en compte de la sonnette.

Si test en attente active ∧ arrivée de la page alors
restaurer le contexte
état = actif
allera continuer l'exécution normale

Ces algorithmes peuvent être réalisés en modifiant le micro-séquenceur du processeur

La solution (h):

- garantit une fréquence à peu près constante de test,
- dégrade légèrement les performances du système.

IV - 7 TEST A INSTANT FIXE

Il serait assez sûr de faire le test à des instants fixés. La fréquence serait ainsi constante. Mais la seule instruction d'interruption (HALT) arrêtant tous les processus un certain temps, c'est impossible. Cela dégraderait considérablement les performances du système et ne tient pas compte de la

structure multiprocesseur. Cela semble peu envisageable dans aucun système multiprocesseur.

IV - 8 TEST EN OISIVETE CONTRAINTE

Il s'agit de forcer le processeur à exécuter un programme de test sans interrompre l'exécution d'un processus.

Pour cela, après un temps T, le processeur ne pourra pas commencer l'exécution d'un processus avant d'avoir été testé complètement.

Si $t \geq T \wedge$ fin de processus alors
état = test
allera début du programme de test

Il faut donc interdire à un processeur de prendre en compte la sonnette après l'instant T.

Le chargement du contexte d'un processus utilise la molécule. Il faut donc mettre un bit d'interdiction de chargement dans le processeur cache par processeur. Le processeur consultera le bit nommé J_i pour le processeur i.

Un compteur C_i par processeur sera nécessaire. Le temps T étant approximatif, ces compteurs ne demanderont pas beaucoup de matériel supplémentaire.

Positionnement de J_i par le processeur cache

Si $(C_i > T)$ alors $J_i \leftarrow 1$

Action du processeur atome

Si fin de test alors $C_i \leftarrow 0$
$J_i \leftarrow 0$; état = oisif

Si (fin du mini-processus) $\wedge$ J_i alors allera début du test.

Le programme de test ne sera pas interruptible. A la fin du test, le processeur se met dans l'état oisif. Ce programme de test sera exécuté par le processeur comme un processus normal, sans état spécial.

Un processeur peut ne finir un processus que très longtemps après T.

Pendant ce temps là les autres processeurs de sa molécule peuvent être oisifs. On pourrait donc prévoir un mécanisme qui commutera les tâches d'un processeur sur un autre processeur oisif de sa molécule au bout d'un certain temps (compté d'une façon très grossière à partir du nombre d'instructions ou autres...). Cela suppose un moyen simple pour réaliser un passage de contexte à l'intérieur d'une molécule.

Le temps T doit être déterminé par simulation. Cette solution :

- permet d'évaluer la fréquence de test en fonction de T,
- permet d'imposer l'exécution d'un programme de test quelle que soit la charge du système,
- diminue les performances du système,
- demande du matériel et une possibilité de passage de contexte dans la molécule.

IV - 9 COMBINAISON DES DEUX SOLUTIONS

La solution (e) ne peut pas être utilisée seule. Avant le temps T, le processeur peut être oisif ou en attente active, et donc testable. Nous combinerons donc les solutions pour obtenir la solution (i).

Un programme de test sera exécuté par les processeurs :

- qui sont oisifs,
- qui sont en attente active,
- après le temps T et la fin du processus.

Ce programme de test sera interruptible dans les deux premiers cas, pas dans le troisième. Le compteur de temps sera remis à 0 dès que, par l'un de ces moyens, la fin du programme sera atteinte.

Le séquenceur devra réaliser les algorithmes définis en IV - 6. De plus le processeur molécule réalisera :

$$\underline{\text{Si}} \quad (C_i > T) \ \underline{\text{alors}} \ J_i \leftarrow 1$$

Les dernières instructions du programme de test seront :

$\underline{\text{Si}}$ fin de test $\underline{\text{alors}}$ $C_i \leftarrow 0$
$J_i \leftarrow 0$
état = oisif.

Pour le 3e cas ($t > T$) on aura

Si (fin du processus) $\wedge J_i$, alors allera contenu du registre de test

Au cas où l'exécution d'un programme de test par un processeur en attente active ne serait pas possible, la fréquence de cette éxécution n'est pas beaucoup diminuée. Mais les performances du système sont diminuées. Cette solution :

- diminue le moins possible les performances du système tout en assurant une fréquence de test évaluable en fonction de T,
- demande un certain matériel supplémentaire.

IV - 10 COMPARAISON DES SOLUTIONS

Nous retiendrons des solutions qui assurent des compromis différents entre la fréquence de test, la diminution des performances du système et le coût.

- Test en oisiveté réelle

  Fréquence : non évaluable,
  différente pour chaque processeur,
  fonction de la charge du système.
  Diminution des performances : nulle,
  Coût : très réduit.

- Test en oisiveté réelle et en attente active

  Fréquence : non garantie,
  à peu près indépendante de la charge du système,
  Diminution des performances : faible,
  Coût : réduit.

- Test en oisiveté réelle, en attente active et en oisiveté contrainte

  Fréquence : évaluable,
  indépendante de la charge du système,
  Diminution des performances : grande quand la charge est grande,
  Coût : important.

V POLITIQUE DE TEST DE L'ENSEMBLE PROCESSEUR CACHE, MEMOIRE CACHE

VI - INTRODUCTION

L'ensemble C = { processeur-cache mémoire-cache } doit être disponible à chaque instant pour les processeurs de la molécule qui sont actifs.

Un processeur actif utilise le processeur cache et il a besoin des informations contenues en mémoire cache ce qui n'est pas possible durant le test de C. Le test de l'ensemble C ne peut se faire qu'en disposant de la molécule complète.

On peut envisager de placer en file d'attente des processus un processus de test, considéré comme processus normal. Cela permettrait de tester l'ensemble C sans matériel supplémentaire.

Cela n'est pas possible:

- il faut un moyen de savoir quels ensembles C ont été testés pour ne pas tester toujours le même et tous les tester,
- un processus ne peut pas réserver tous les atomes d'une molécule.

On pourrait aussi faire exécuter un programme de test à des instants fixés. Cela n'est pas possible : il faudrait pouvoir interrompre les processeurs, ce qui n'est pas prévu.

L'ensemble C ne peut être testé que lorsque tous le sprocesseurs sont oisifs.

- soit oisiveté réelle, ce qui ne se produit que pour une faible charge du système,
- soit oisiveté contrainte, en interdisant aux processeurs de prendre en compte les appels de la sonnette à partir d'un temps T'.

V - 2 TEST EN OISIVETE REELLE

Tous les processeurs de la molécule ne seront oisifs en même temps que rarement.

A ce moment, un programme de test sera exécuté. Il sera non interruptible,

pour profiter de l'évènement. A la fin de l'exécution du programme de test de l'ensemble C, les processeurs atomes seront tous oisifs. Donc la molécule se remettra en état de **test**, à moins qu'à cet instant précis il y ait appel de la sonnette. Il faut fixer une fréquence maximum de test, sinon une molécule oisive resterait en état de test jusqu'à la fin des temps...

Il faut donc au bout du temps M (temps minimum entre deux tests de C), mettre à 1 une bascule B qui autorise le test.

Si t = M alors B ← 1 ;

Le test s'effectuera donc si la molécule est oisive et si B est à 1. La molécule est oisive si les bascules "état test en oisiveté" T_i de tous les processeurs sont à 1.

Si $T_1 \ldots T_i \ldots T_n \ldots B$ alors allera initialisation du test de C.

Le fait que le test est non interruptible a peu de conséquences sur les performances du système : le degré de parallélisme entre les processus est peu diminué et la durée d'exécution d'un programme de test est de l'ordre de celle d'un processus.

Cette solution

- diminue peu les performances du système,
- n'assure qu'une faible fréquence de test pour C.
- est compatible avec la solution f de test des atomes.

Il faut garantir une plus grande fréquence de test pour les autres solutions de test du processeur atome.

V - 3 TEST EN OISIVETE CONTRAINTE

Si l'ensemble C n'a pas été testé depuis un temps T', on interdit aux processeurs de prendre en charge d'autres processus, après qu'ils aient fini celui en cours d'exécution. Les processeurs rendus ainsi oisifs sont mis en état de test en oisiveté. Quand tous les processeurs sont oisifs, on commence le test de l'ensemble C. Le bit d'interdiction de prise en compte s'appelle K. Le registre N sera à n si les n processeurs de la molécule sont oisifs.

Les algorithmes de mise à l'état "test en oisiveté" et en "attente active" sont les mêmes, ainsi que leur interruption.

Le processeur cache réalisera :

Si (t ≥ T') alors K ← 1

A la prise en compte d'un processus le processeur atome réalisera :

Si K alors si N = n-1 alors initialiser test de l'ensemble C
appel des autres processeurs
sinon N ← N + 1
état = test en oisiveté
allera contenu du registre de test
Sinon prise en compte de l'appel

A la fin du test de l'ensemble C, le processeur cache réalisera

Si fin de test de C alors K ← 0
N ← 0
Compteur ← 0

Les processeurs atomes se mettront dans l'état oisif.

Cette solution permet d'utiliser l'oisiveté forcée des processeurs atomes pour qu'ils se testent. Ce test étant du mode : test en oisiveté, le dernier processeur à devenir oisif l'interrompt.

En effet, des processeurs en état de test font appel à la mémoire cache, ce qui est impossible si C est en cours d'exécution d'un programme de test.

Quand C sera en test, ou bien on utilise les processeurs atomes pour le test de C ; ou bien on les fait boucler sur une instruction sans appel de l'ensemble C.

Cette solution :

- garantit une bonne fréquence de test de C,
- diminue les performances du système moyennement,
- respecte la structure du système,
- demande un peu de matériel supplémentaire.

VI - CONCLUSION

Nous avons présenté une série de solutions représentant des compromis différents entre fréquence du test (donc disponibilité) et coût en matériel et en performances.

- test des processeurs atomes en oisiveté,
  test de l'ensemble C quand tous les processeurs atomes sont oisifs

- test des processeurs atomes en oisiveté et en attente active
  test de l'ensemble C quand les processeurs atomes sont en oisiveté imposée avec T' grand

- test des processeurs atomes en oisiveté, en attente active, dès que possible, après T
  test de l'ensemble C quand les processeurs atomes sont en oisiveté imposée avec T' petit

La fréquence de test croît dans l'ordre de ces solutions, ainsi que le coût.

Les temps T et T' devront être évalués par simulation.

CHAPITRE V

APPLICATION :

LE CENTRAL TELEPHONIQUE E10

INTRODUCTION

En I,la puissance moyenne d'un système distribué modulaire,le centre E10, est évaluée.En II, une politique de détection en cours de fonctionne--ment est proposée pour un sous système de E10: le GUR.

I - EVALUATION DE LA PUISSANCE MOYENNE DE E10

I - 1.PRESENTATION DU SYSTEME.

Un centre E10(fig.V-1) est un système distribué modulaire, composé de 5 sous-systèmes:

- le multienregistreur MR, composé de 8 modules,
- le traducteur TR composé de 2 modules,
- le marqueur MQ composé de 2 modules,
- le taxeur TX non modulaire,
- le réseau de connexion RCX.

Les GUR réalisent l'interface du central avec les abonnés.

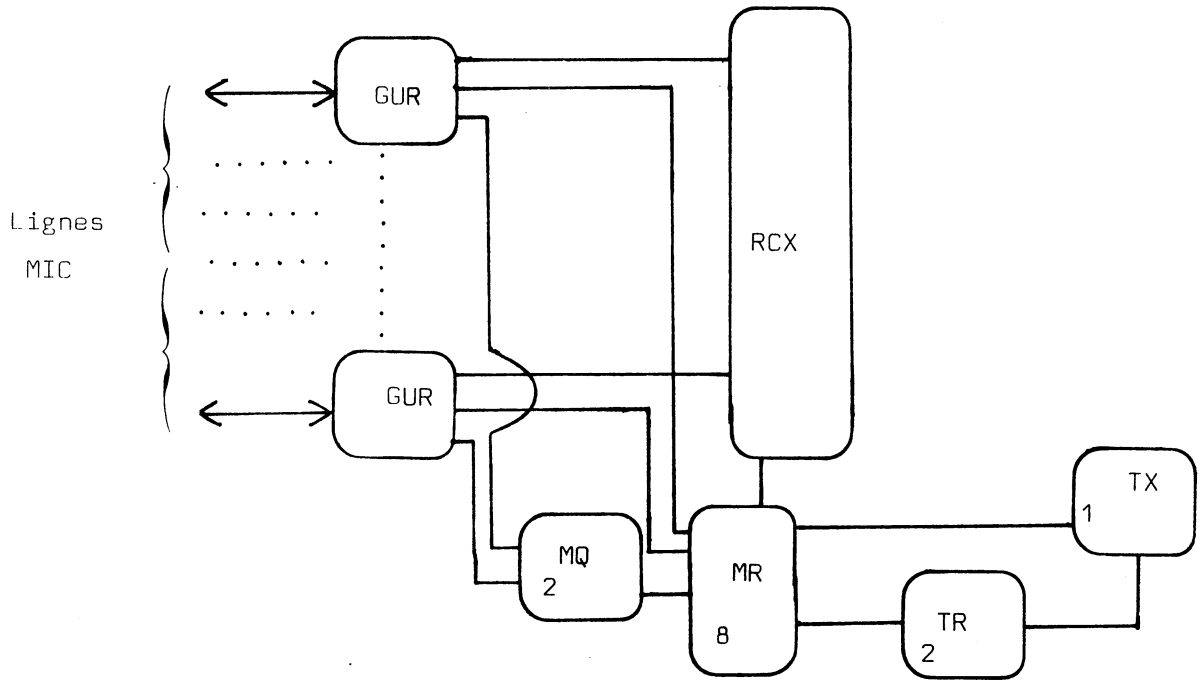


Figure V-1.

I - 2. FIABILITE DES MODULES.

Les modules ont un taux de pannes proportionnel à la quantité de matériel qui les réalisent, à l'exception du RCX,qui est réalisé à l'aide de composants plus fiables (24). On obtient le tableau suivant:

SOUS-SYSTEME	MR	TR	MQ	TX	RCX
NOMBRE DE MODULES	8	2	2	1	1
λT D'UN MODULE DU SOUS-SYSTEME	10^{-3}	10^{-3}	10^{-2}	10^{-4}	10^{-4}

I - 3. PUISSANCE DU SYSTEME.

La courbe de la figure V-2 represente la μ-fiabilité $R_T(\mu)$ du système.

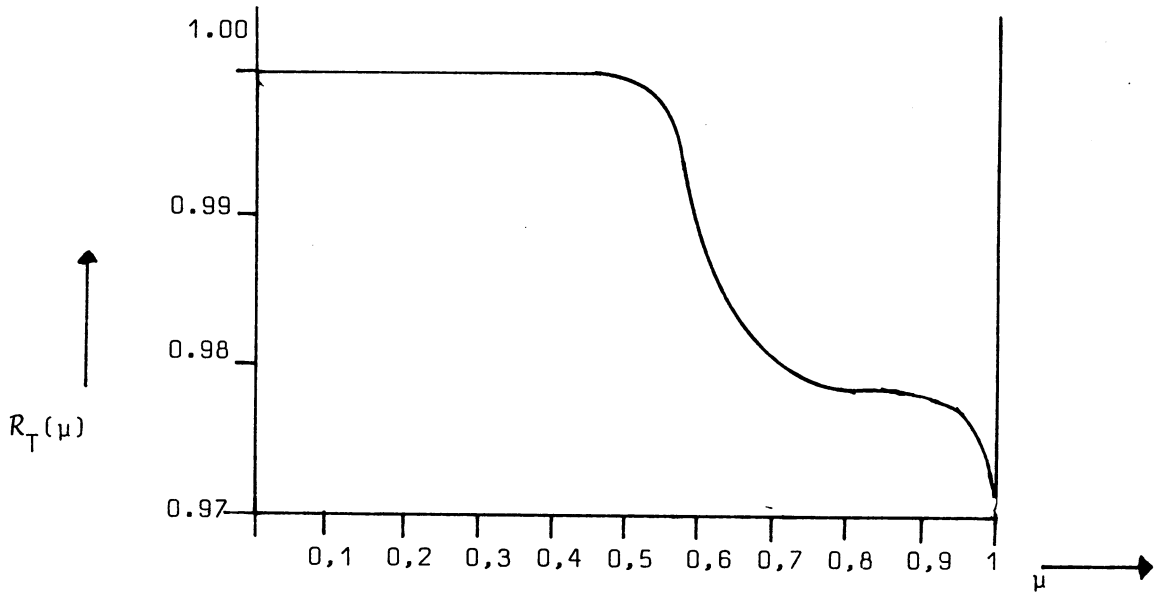


Figure V-2

Le calcul de l'espérance de la puissance normalisée donne :

$P_{ex}(T) = 0,988$

avec un écart-type de 0,006.

Donc le centre E10 travaille au temps T en moyenne à 0,988 et au minimum à 0,982 de sa puissance maximale.

II - DETECTION EN COURS DE FONCTIONNEMENT : LE GUR.

Nous étudieront plus particulièrement un sous-système du centre E10 : le GUR ou Groupe d'Unités de Raccordement, présenté en (22), en vue de déterminer une politique de détection en cours de fonctionnement.

II - 1. PRESENTATION DU GUR.

Le GUR réalise l'interface entre 32 multiplex MIC (soit 16 unités de raccordement)et le reste du central (marqueurs, multi-enregistreur, réseau de connexion, taxeur,etc..). Il assure la synchronisation des liaisons entrantes, le prétraitement de la signalisation, la gestion et la surveillance des circuits, la défense et la maintenance. Il est relié aux autres sous-systèmes du central par des liaisons spécialisées pour chaque échange possible.

Il est composé de :

- 2 unités centrales UC_1 et UC_2.
- 2 modules d'échange MLE_1 et MLE_2.
- 16 modules de raccordement MRX_1 à MRX_{16}.
- 4 bus $BRME_1$ et $BRME_2$ (en entrée des UC) , $BRMS_1$ et $BRMS_2$ (en sortie des UC).
- 2 liaisons de commande de modules LCM_1 et LCM_2 , et de signalisation des fautes LSF_1 et LSF_2 .

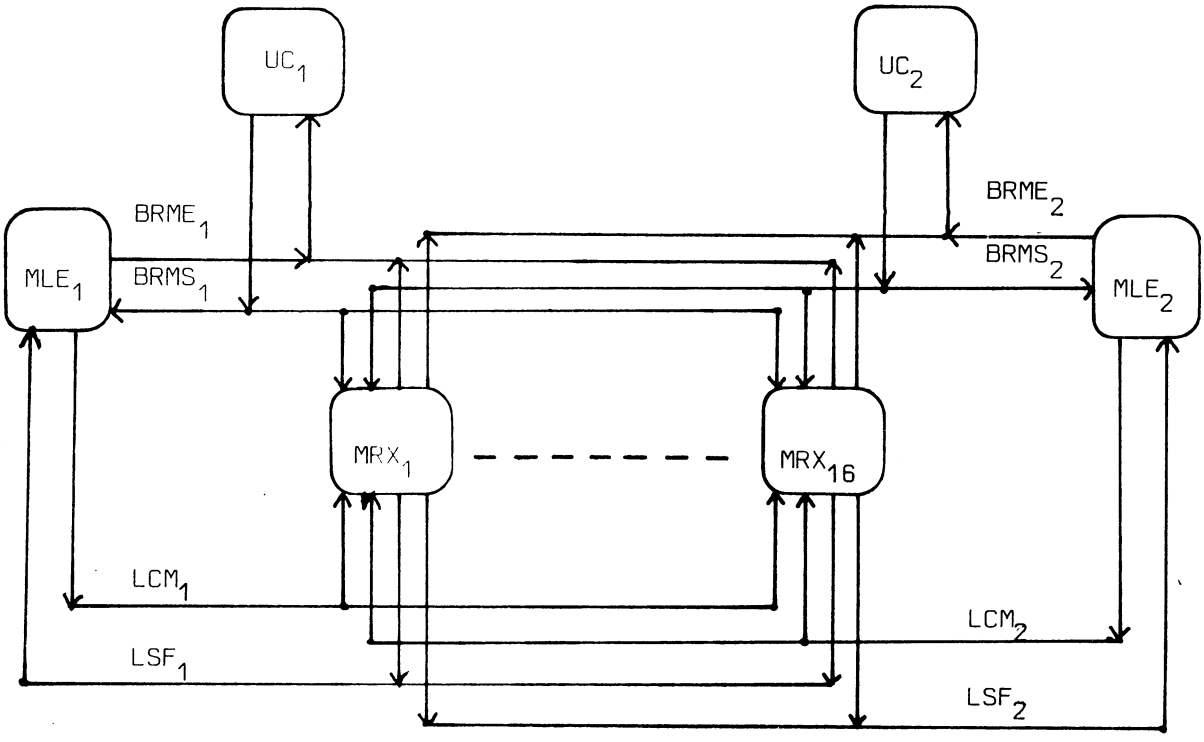


Figure V-3

a) les unités centrales (UC) ou modules de commande programmés (MCP).

Ce sont des processeurs microprogrammés travaillant sur des octets avec des instructions de 16 bits. Les UC ne sont pas interruptibles : elles exécutent un grand nombre de tâches, chaque tâche étant de tres faible durée ($\leq$ 1ms) , ce qui leur permet de travailler en temps réel.La prise en compte des évènements extérieurs se fait par exploration systématique des différents modules gérés par l'UC. Le processus exécuté par l'UC est inscrit dans une ROM appelée mémoire de programme (Cf. paragraphe II - 3.).

Les deux UC sont identiques matériellement et logiciellement.

b) les modules d'échange (MLE) .

Chacun des modules d'échange MLE_1 et MLE_2 est affecté d'une manière rigide à $l'UC_1$ et à $l'UC_2$. Un module d'échange est un ensemble d'automates câblés qui réalisent l'interface entre l'UC à laquelle le MLE est affecté et le reste du système E10 (marqueurs en particulier) . Un MLE est relié à tous les modules de raccordement par les liaisons LCM et LSF .

c) les modules de raccordement (MRX) .

Le type des modules dépend de la liaisons multiplex à raccorder.Cepen--dant, certaines de leurs fonctions sont indépendantes du type de modules. Ce sont les fonctions non liées à la signalisation, et pour ce qui nous intéresse, les liaisons avec les UC et les MLE.

d) les bus

Chaque UC est reliée à son module d'échange et à tous les MRX par deux bus unidirectionnels; le bus BRMS va de l'UC vers les modules, BRME va des modules vers l'UC.

e) les liaisons LCM et LSF

Un MRX est affecté à une UC et une seule par les liaisons de commande des modules LCM, positionnée chacune par le MLE correspondant, sur ordre du reste du système.

La liaison de signalisation de faute LSF est positionnée par les MRX et signale au MLE la présence d'une faute.

II - 2 PROPRIETES DE TOLERANCE AUX PANNES.

Ce système ne nécessite pas une haute sécurité : de par sa fonction il peut tolérer quelques erreurs avant qu'une panne ne soit détectée; en effet il traite des informations redondantes temporellement.

Par contre, ce système doit avoir une haute disponibilité; pour cela il dispose de possibilités de reconfiguration après detection d'une panne ou d'une erreur.

a) possibilités de reconfiguration

Au point de vue de la reconfiguration, le GUR est composé de trois ensembles :

- un ensemble E_1 comprenant UC_1, MLE_1, $BRMS_1$, $BRME_1$, LCM_1 et LSF_1
- un ensemble E_2 comprenant UC_2, MLE_2, $BRMS_2$, $BRME_2$, LCM_2 et LSF_2
- les modules de raccordement.

Les deux premiers ensembles sont identiques et interchangeables. En effet, le trafic peut être traité dans sa totalité par l'un seulement des ensembles. En l'absence de pannes, les 8 premiers modules MRX_1 à MRX_8 sont affectés à l'UC_1; les 8 derniers (MRX_9 à MRX_{16}) sont affectés à l'UC_2. Après détection d'une panne dans un ensemble, E_1 par exemple, le module d'échange MLE_2 affecte les modules MRX_1 à MRX_8 à l'UC_2, à l'aide de la liaison de commande LCM_2.

Cette reconfiguration se fait sans perte d'informations et sans pertu-bations; en effet toutes les informations relatives aux circuits et aux commmunications sont stockées dans les MRX.

Les modules de raccordement ne peuvent être reconfigurés; mais la panne d'un module de raccordement n'affecte qu'une partie des abonnés.

b) détection discontinue

La détection des pannes et des erreurs se fait à l'aide des deux techniques décrites aux chapitres III et IV.

Le test des ensembles E_1 et E_2 se fait en oisiveté contrainte : à des intervalles de temps fixés tous les MRX sont affectés à une UC, par exemple UC_2; l'UC_1 exécute alors un écoulement spécialisé de test et teste

l'ensemble E_1.

Le basculement du trafic sur une seule UC est effectué sans perte de performance (une seule UC est capable de traiter la totalité du trafic) et sans perte d'information (les informations sont stockées dans les MRX) .

c) détection continue

Un certain nombre de vérifications sont effectuées au cours de phases de vérification (voir paragraphe suivant). En particulier, chaque MRX peut signaler aux deux MLE, par l'intermédiaire des liaisons LSF, des erreurs commmises par l'une des UC.Trois types de fautes sont signalés:
- LSF_1 et LSF_2 positionnées à 01 signifie que les liaisons LCM affectent le module aux 2 UC simultanément.
- LSF=00 signale que le module n'est affecté à aucune UC.
- LSF=10 signale que le module n'a pas été exploré par l'UC à laquelle il est affecté depuis plus de 32 ms;ce contrôle de type watch-dog permet de détecter une erreur de séquencement de l'UC en cause.

II - 3. MODELISATION DU PROCESSUS EXECUTE PAR LES UC.

Chaque UC a trois types de sous-processus à effectuer :
- routine interne
- exploration du MLE
- exploration des différents MRX

La valeur d'un index K, variant de 0 à 32, détermine quel sous-processus doit être exécuté;si K est impair,exploration du MLE;si K est pair exploration du MRX de n° K/2;si K=0 exécution de la routine interne, qui comprend un test rapide.L'index K est incrémenté à chaque passage dans la phase initiale I.

La valeur de K/2 variant de 1 à 16, une UC tente d'explorer tous les MRX. Seuls les MRX qui lui sont affectés répondent à son appel.Cela permet de ne pas modifier le processus après basculement du trafic sur une seule UC (dû au test ou à la reconfiguration).

Le réseau représentant le processus est un réseau pré-structuré (fig. V-4)

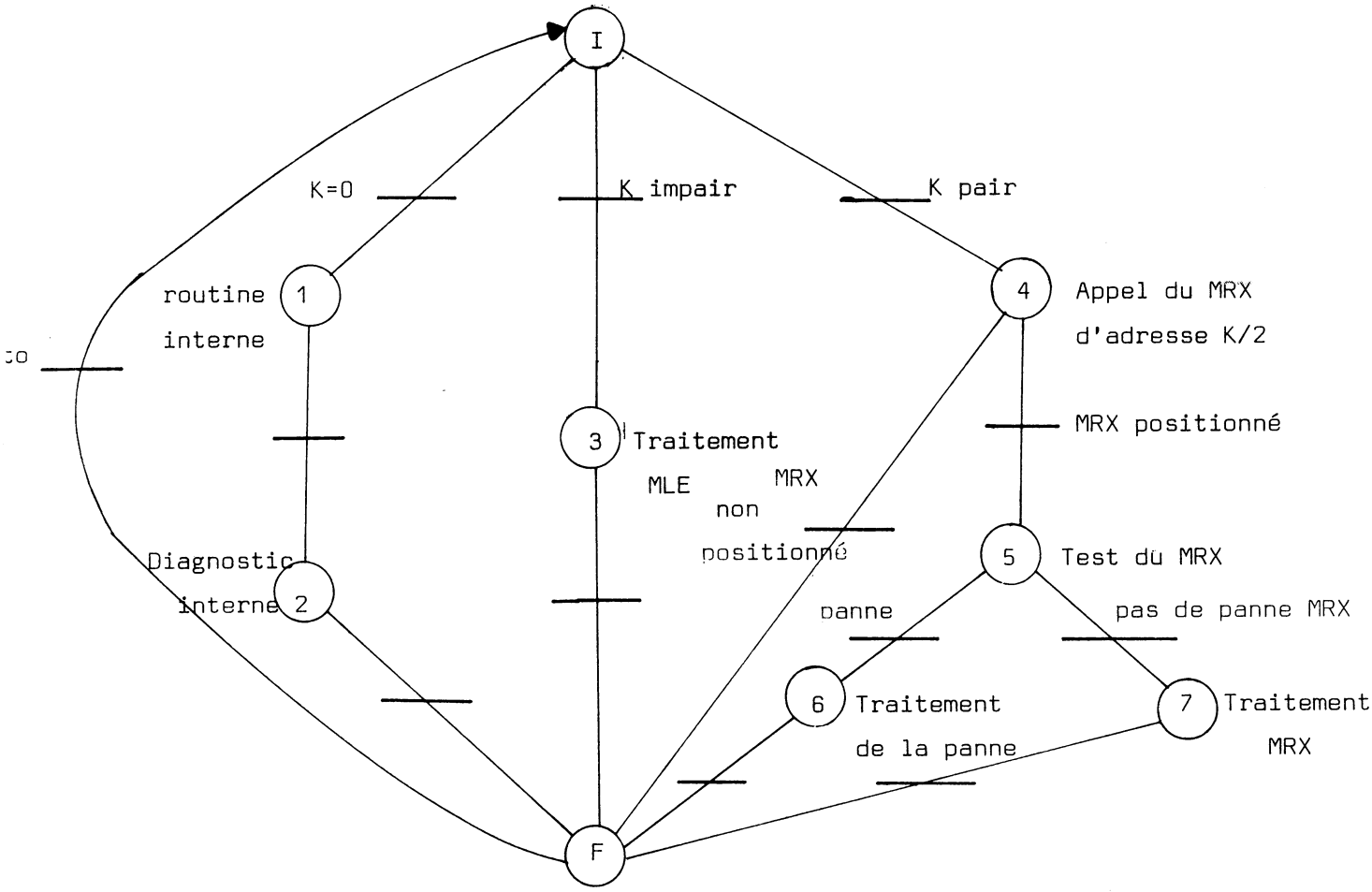


Figure V-4

Les phases I et F sont les phases initiale et finale.Les phases 2,5 et 6 sont des phases de vérification : la phase 2 vérifie le fonctionnement correct de l'UC par des vérifications fonctionnelles; la phase 5 vérifie le bon fonc--tionnement du MRX appelé; la phase 6 traite une panne du MRX et la signale à l'extérieur.

La phase 3 est une phase de contamination : le processus effectué par les UC peut contaminer le MLE . La phase 7 est une phase dangereuse : l'UC, en traitant les informations contenues dans le MRX peut communiquer des informations fausses au monde extérieur (ici l'abonné).La phase 7 est précédée d'une phase de vérification, ce qui assure une certaine sécurité.

La phase 4 est une phase observable par le MRX; elle sert à établir le watch-dog.La partie du processus du MRX réalisant le watch-dog est représenté fig.V- 5.

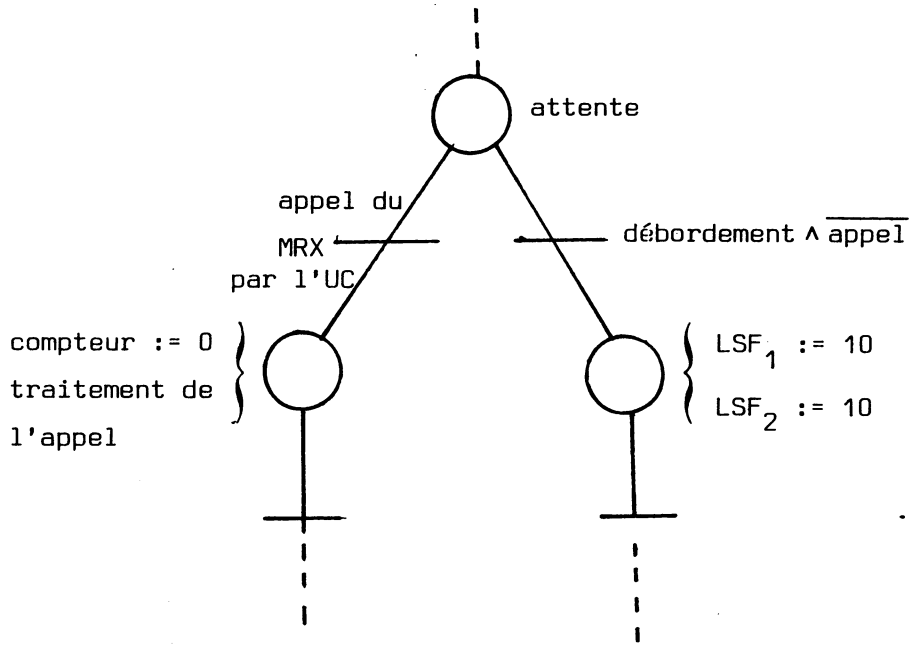


Figure V-5

Chaque MRX dispose d'un compteur qui déborde dès que le temps entre deux appels de l'UC est supérieur à 32 ms. Ce compteur est remis à 0 à chaque appel de l'UC.

Ce système utilise donc les deux possibilités de test en cours de fonc-tionnement, ce qui assure une bonne détection des pannes, une certaine sécurité et une grande disponibilité.

REFERENCES

1) J.C. LAPRIE:"Prévision de la sûreté de fonctionnement et architecture de structures numériques temps réel réparables".Thèse d'Etat,Université Paul Sabatier,16 juin 1975,Toulouse.

2) J.L. PAUL:"PEGASE, programme d'évaluation globale par analyse de la simulation des erreurs".Thèse de 3e cycle,Université Paul Sabatier,16 juin 1975,Toulouse.

3) K.M. CHANDY, C.V. RAMAMOORTHY, A. COWAN: "A framework for hardware-software tradeoffs in the design of fault-tolerant computers". Proc. of Fall Joint Computers Conference.1972,pp 55 à 62.

4) D.A. ANDERSON,G. METZE:"Design of totally self-checking check circuits for m-out-of n codes".IEEE Trans. on Computers,Mars 1973.

5) J.F. WAKERLY:"Partially self-checking circuits and their use in performing logical operations". Int. Fault Tolerant Computing Symp., Palo Alto,1973.

6) W.C. CARTER, K.A. DUKE, D.C. JESSEP :"A simple self testing decoder checking circuit". IEEE Trans. on Computers.C 20, 1971,pp 1413 à 1414.

7) M.D. BEAUDRY:"Performance related measures for computing systems". Proc. of Fault Tolerant Computing Symposium.Los Angeles,Juin 1977,pp 29 à 34.

9) P.D. WELCH:"On the reliability of polymorphic systems".IBM Systems Journal,vol.4,n° 1, 1965,pp 43 à 52.

10) Z.AVIZIENIS, B. PARHAMI :"A fault tolerant parallel computer system for signal processing".Proc. of Fault Tolerant Computing Symposium.Urbana, 1974,pp 2-8 à 2-19.

11) G. GERMAIN, F. BROWAEYS, C. MERAUD :"COPRA ,un multiprocesseur à haute sûreté de fonctionnement".Doc. interne EMD-SAGEM.

12) J.S. MILLER :"Design features of an aerospace multiprocessor".Int. Workshop on Computers Architecture,Grenoble,1973.

13) M.A. FISCHLER, O. FIRSCHEIN :"A fault tolerant multiprocessor archi-
-tecture for real-time control applications". Proc. of 1st Annual Symp.
on Computer Architecture,Floride 1973 ,pp 151 à 160.

14) M. ZACHARIADES:" MAS : Réalisation d'un langage d'aide à la description et à la conception des systèmes logiques", Thèse de 3e cycle INPG, Septembre 1977.

15) C.A. PETRI :"Communication with Automata".Supplement 1 to Technical Report RAD C-TR-65-337,1,Griffiss Air Force Base,New York, 1966.

16) R. VALETTE :"Sur la description,l'analyse et la validation des systèmes de commande parallèles". Thèse d'Etat,Université Paul Sabatier,24 novembre 1976,Toulouse.

17) P. CASPI,C.ROBACH:"Modeles pour l'etude de la testabilité des systèmes informatiques". Congrés AFCET 1977 , Versailles ,novembre 1977.

18) C. ROBACH,G. SAUCIER :"Diversified test methods for local control units" IEEE Trans. on Computers.Vol C24 n°5,Mai 1975,pp 562 à 566.

19) C. ROBACH, G.SAUCIER:" System modeling and diagnosability".COMPCON Spring 77,San Francisco,MARS 1977,pp 294 à 299.

20) C. BELLON, G. SAUCIER:"On line test modeling in non-redundant distributed systems".Proc. of Fault Tolerant Computing Symposium,Los angeles,Juin 1977 pp 94 à 102.

21) C. ROBACH:"Méthodologie de test de processeurs.Impact sur la conception". Thèse de Docteur-Ingénieur,USMG,mai 1975,Grenoble .

22) D. FEURSTEIN, M.SERVEL:"Système E10:Groupe d'unités de raccordement". Bulletin technique d'information XVII 2,1975,pp 37 à 52.

23) Y.DESWARTE:"Structure Multimicroprocesseur à haute sécurité".Rapport Technique du lot 4 du contrat DRME 73/373.

24) B.R. BORGERSON, R.F. FREITAS :"An analysis of PRIME using a new relia-
-bility model".Proc of Fault Tolerant Computing Symp.,Urbana juin 1974,
pp 2-26 à 2-31.

25) C. BELLON, G. SAUCIER : "Graceful degradation of performance in fault tolerant multiprocessor system". Euromicro Symposium, Venise, Octobre 1976.

26) C. BELLON, G. SAUCIER : "Modularity in fail softly distributed systems", Euromicro Symposium, Amsterdam, Octobre 1977.

dernière page de la thèse

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 Avril 1974,

VU le rapport de présentation de :

- Madame G. SAUCIER, Maître de Conférences à l'Institut National Polytechnique de Grenoble

Mademoiselle Catherine BELLON

est autorisée à présenter une thèse en soutenance pour l'obtention du titre de DOCTEUR de TROISIEME CYCLE, spécialité "Génie Informatique".

Grenoble, le 20 Juillet 1977

Ph. TRAYNARD
Président
de l'Institut National Polytechnique