Migration et points de reprise dans les systèmes distribués

Techniques de mise en œuvre et mécanismes communs

Gilles Muller* **et Pierre Sens+**

* *IRISA/INRIA*
Campus de Beaulieu
35042 Rennes Cedex
E-mail : Gilles.Muller@irisa.fr

\+ *MASI*
4 place Jussieu
75252 Paris Cedex 05
Email : Pierre.Sens@masi.ibp.fr

RÉSUMÉ. *La migration de processus et les points de reprises sont deux mécanismes qui améliorent le service offert par le système d'exploitation pour l'exécution des applications utilisateur. Ces deux mécanismes reposent sur des techniques d'implémentation voisines et ont été parfois intégrés de manière conjuguée au sein de systèmes distribués. Ce document présente une synthèse des techniques de mise en œuvre et des systèmes existants. Plus particulièrement, nous mettons l'accent sur les techniques et problèmes communs de mise en œuvre.*

MOTS CLÉS : *Points de reprise, tolérance aux fautes, migration, systèmes distribués, transparence.*

1 Introduction

La migration de processus et la tolérance aux fautes sont deux mécanismes qui améliorent le service offert par le système d'exploitation pour l'exécution des applications utilisateur. La migration de processus permet la modification dynamique du placement d'un processus pour utiliser au mieux les ressources disponibles dans le système ; la tolérance aux fautes assure la poursuite de l'exécution de l'application après la défaillance d'un ou plusieurs processeurs (ou machines) de l'architecture.

Parmi les différentes techniques de tolérance aux fautes, nous nous intéressons plus particulièrement à la sauvegarde de points de reprise et au retour arrière. En cas de défaillance d'un processeur, le ou les processus s'exécutant précédemment sur ce processeur doivent être relancés sur une machine de secours. On peut donc assimiler le recouvrement du processus à une migration non anticipée de la machine défaillante vers la machine de secours. Toutefois, une différence majeure entre la migration et le recouvrement est que le dernier état du processus est perdu. Il est par conséquent nécessaire de reprendre l'exécution à partir d'un état précédent du processus (c.a.d. le point de reprise). Il en résulte évidemment une perte de calcul.

On comprends aisément que la migration et le retour arrière reposent sur des techniques de bases similaires où voisines. Il est à noter que certains systèmes tels que Condor [LIT 92], GatoStar [FOL 95] et Stardust [CAB 96] tirent profit de cette similarité pour implémenter migration et points de reprise. Ce document présente une synthèse de ces techniques en exhibant leurs points communs et leurs différences. Nous étudions en premier dans la section 2, les différentes techniques de création de points de reprise. Plus précisément, dans le cadre d'applications paralèlles scientifiques composées de processus communicants, une des difficultés majeures est de maintenir à tout moment un état cohérent. Nous décrivons les deux principales approches reposant, soit sur des points de reprise coordonnés, soit sur des points de reprise indépendants et la journalisation de messages.

Dans la section 3, nous présentons ensuite les techniques de migration qui ont été proposées dans le cadre de systèmes d'exploitation distribués. Nous présentons les différentes solutions permettant de migrer l'état du processus, leur impact sur les performances et les problèmes de sensibilité du système vis à vis des défaillances.

La section 4 est dévolue aux problèmes communs d'implémentation de la migration et des points de reprise. Plus précisément, nous décrivons les aspects liés à la transparence. Dans la section 5, nous présentons les performances que l'on peut attendre des implémentations existantes. Nous concluons en section 6.

2 Points de reprise et retour arrière

Le recouvrement de faute à partir de points de reprise est une technique très couramment utilisée pour la tolérance aux fautes matérielles des applications réparties [STR 85, ELN 92, XU 93, MUL 94]. Le système sauvegarde périodiquement, dans des points de reprise, l'état des processus sur un support stable pour pouvoir relancer les processus en cas de faute.

Dans les systèmes centralisés, le recouvrement à partir de points de reprise est simple à réaliser. Périodiquement, le *processeur* sauvegarde son état sur support stable. Lorsqu'une faute survient, le processeur reprend son exécution à partir de son dernier point de reprise. En revanche, dans un système réparti, la gestion des reprises s'avère être nettement plus compliquée. En effet, à la différence d'un système centralisé, les retours en arrière affectent uniquement une partie du système pouvant entraîner un état global incohérent. Les techniques de recouvrement à partir de points de reprises sont divisées en deux classes principales : les points de reprise indépendants et les points de reprise coordonnés. Dans la première approche, les processus réalisent indépendamment leurs points de reprise et se coordonnent lors de la phase de recouvrement. Dans la seconde approche, les processus se coordonnent au moment du point de reprise de manière à faciliter la reprise.

Nous présentons tout d'abord les techniques de création de point de reprise pour les processus simples. Ensuite, nous étudions les deux méthodes coordonées et indépendantes.

2.1 Point de reprise d'un simple processus

Un point de reprise est l'image de l'état local d'un processus. Il regroupe les informations suivantes :

- le contexte mémoire. Il s'agit de l'espace d'adressage (virtuel) utilisé par le processus et des différents segments qui le compose. Dans un système de type Unix, le contexte mémoire regroupe au minimum un segment pour le code, un segment pour la pile et un segment pour les données et le tas.

- le contexte processeur. Il s'agit des différents registres du processeur. Par nature, les procédure de gestion du contexte processeur sont dépendantes de l'architecture et doivent être réécrite lors d'un portage.

- le contexte système. Ce sont toutes les données qui caractérisent le processus au sein du système d'exploitation. De façon non exhaustive, nous pouvons citer les fichiers ouverts, les communications réseaux, les signaux ... Dans les systèmes à points de reprise orientés pour le support d'application de calcul parallèle, il est souvent supposé que seuls les fichiers standard (i.e., entrée, sortie, erreur) sont utilisés ; le reste du contexte système est négligé.

La sauvegarde de l'état local du processus peut être optimisée soit en réduisant le volume de données à sauvegarder, soit en évitant de bloquer le processus lors de l'établissement du point de reprise.

Réduction du volume des données à sauvegarder

Afin d'éviter de sauver la totalité du contenu des segments mémoire lors d'un point de reprise, il est possible de ne sauvegarder que les pages modifiées depuis le dernier point de reprise. Cette technique appelée sauvegarde *incrémentale* a été introduite dans le système Targon [BOR 89] et employée par Elnozahy *et al.* [ELN 92] et dans le système FTM [MUL 94]. Pour pouvoir être utilisée, cette méthode nécessite de pouvoir accéder à la gestion des défauts de page. Une variante de cette technique maximisant le nombre de pages modifiées a été également décrite par Cabillic *et al.* dans le contexte d'une mémoire partagée distribuée recouvrable [CAB 95].

Réduction de la durée de gel du processus

Afin de réduire la durée du gel du processus, l'idée de base consiste à anticiper la recopie de l'état mémoire avant le blocage du processus. Cependant, si un processus modifie une de ses pages avant le blocage, l'image sauvegardée ne représente alors plus un réel état du processus. Il y a deux solutions à ce problème.

La première solution utilise le mécanisme interne du système qui copie les pages en cas d'écriture (*copy-on-write*). Au début du point de reprise, les pages à sauvegarder sont protégées en écriture ; cette protection est retirée dès que les pages sont écrites sur support fiable. Si un processus veut modifier une des pages protégées, le système copie la page dans une page mémoire nouvellement allouée et retire la protection sur la page à modifier. La nouvelle page allouée n'est pas accessible du processus ; elle est utilisée uniquement par le protocole de création du point de reprise. Cette technique est employée par Li *et al.* [LI 90] et Elnozahy *et al.* [ELN 92].

La seconde solution repose sur une itération du processus de recopie [JOH 89]. Cette technique qui a été initialement développée pour la migration [THE 85] consiste en une recopie des pages dans une zone mémoire hors de l'espace d'adressage du processus. Ainsi, elles peuvent être écrites sur le support fiable sans interrompre l'exécution du processus. Celui-ci peut modifier n'importe quelle page de son espace. Si le nombre de pages modifiées est supérieur à un certain seuil, le coût de la recopie est estimé trop important et le processus reste bloqué durant le point de reprise. La méthode de recopie évite les traitements coûteux et complexes du copy-on-write, mais elle peut devenir bloquante si trop de pages sont modifiées. Elnozahy *et al.* ont intégré ces deux techniques dans le système V et ont mesuré que le temps pour réaliser un point de reprise avec la méthode de recopie est en moyenne 40% plus grand qu'avec le copy-on-write.

2.2 Points de reprise pour un ensemble de processus communicants

Nous présentons les méthodes de recouvrement pour les applications composées de processus communicants. Le but de la reprise est de reprendre l'exécution à partir d'un état cohérent. Nous présentons tout d'abord la notion de cohérence dans un système réparti. Puis, nous décrivons les méthodes de recouvrement reposant sur des points de reprise coordonnés puis celles utilisant des points de reprise indépendants.

2.2.1 États globaux cohérents dans les systèmes répartis

La notion d'état global cohérent est un point clef des systèmes répartis. Elle a été traitée par [RAN 75, RUS 77, POW 83b] et formalisée par Chandy et Lamport [CHA 85]. Nous résumons ici leurs définitions.

Dans le modèle d'exécution répartie, un événement est soit un changement d'état spontané d'un processus, soit l'envoi ou la réception d'un message. Nous pouvons définir une relation de causalité entre les événements. Un événement a précède un événement b si et seulement si :

1) a et b sont des événements du même processus et a a lieu avant b ; ou

2) a est l'envoi d'un message m par un processus et b est la réception de m par un autre processus ; ou

3) a précède c et c précède b.

L'état local d'un processus P est défini par l'état initial de P et la suite des événements ayant eu lieu dans P. *L'état global* d'un système est l'ensemble d'états locaux de tous les processus et des canaux de communication. L'état des canaux de communication est l'ensemble des messages envoyés et non encore reçus. Un état global est cohérent si tous ses événements respectent la relation de causalité.

Nous pouvons décrire l'occurrence des événements dans le temps par un diagramme temporel dans lequel les lignes horizontales représentent le temps des processus, les points les événements et les flèches les messages. Dans cette représentation, un état global est une courbe divisant le diagramme en deux. L'état des canaux est l'ensemble des flèches (messages) traversant cette courbe. La figure 1 représente un diagramme temporel pour un système de quatre processus.

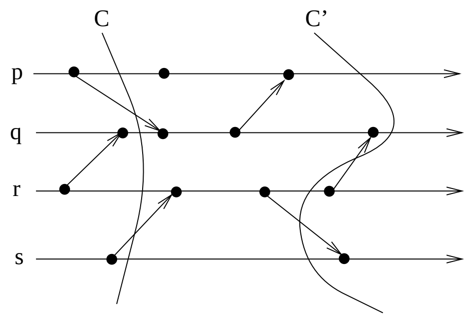

FIG. 1 – *Courbes d'état global*

Une courbe d'état global est *cohérente* (i.e., représente un état global cohérent) si aucune flèche ne commence en partie droite et ne se termine en partie gauche. Cette définition de la cohérence illustre le fait qu' *un message ne doit pas être reçu avant d'être envoyé.* Par exemple, les courbes C et C' sont respectivement cohérente et incohérente. Une courbe consistante est *une ligne de recouvrement.*

Un ensemble de points de reprise est cohérent si les états sauvegardés forment un état global cohérent. Pour obtenir un système de nouveau cohérent, il faut reprendre les processus à partir de la dernière courbe d'état cohérente.

Ainsi, la reprise d'un processus peut entraîner une avalanche de reprises de processus valides. Dans le pire des cas, tous les processus peuvent être repris à partir de leur état initial [RAN 75, RUS 77]. La figure 2 illustre ce phénomène d'*effet domino* (les 'x' représentent les points de reprise).

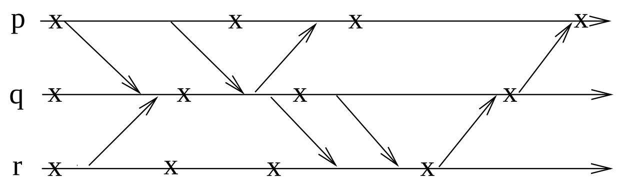

FIG. 2 – *Effet domino*

Si le processus p défaille, il doit être repris à partir de son dernier point de reprise. Dans ce cas, les autres processus ne sont pas concernés par la reprise. Si le processus q défaille, il doit être repris à partir de son dernier point de reprise. Or, ce point précède une interaction avec le processus r ; ce dernier doit alors être repris au premier point précédent cette interaction. En suivant ce principe, si le processus r défaille, tous les processus doivent reprendre leur exécution à partir de leur premier point de reprise.

2.2.2 Points de reprise coordonnés

Dans cette approche, les processus se synchronisent de façon à ce que l'ensemble des points de reprise représente à tout moment une ligne de recouvrement (i.e., un état global cohérent) [CHA 85]. A la suite d'une faute, les processus d'une même application sont repris uniquement à partir de leur dernier point de reprise limitant ainsi l'effet domino [SIL 92]. Les principaux inconvénients de ces techniques sont le surcoût entraîné par les protocoles de synchronisation et la charge importante du support stable durant la réalisation d'un point de reprise [TAM 84, KOO 87, LEU 88, KAA 92, MUL 94]. Pour minimiser les synchronisations, plusieurs travaux se sont concentrés sur la façon de réduire le nombre de processus participant à un point de reprise [KOO 87, LEU 88, ISR 89, MUL 94] ou de diminuer le nombre de messages nécessaires à la synchronisation [BRI 84, BHA 88, TON 89, CRI 91].

L'algorithme de base consiste à geler tous les processus pendant la réalisation du point de reprise. Il faut alors attendre que tous sauvegardent leur état pour continuer l'exécution.

Koo et Toueg propose une amélioration significative de cet alogrithme. Lorsqu'un processus p décide de réaliser un point de reprise, il propage sa demande uniquement aux processus suceptibles de créer une incohérence en cas de reprise à savoir l'ensemble des processus dont il a reçu un message. Chaque processus recevant une telle demande ne réalise par forcément un point de reprise, il teste tout d'abord si il est toujours en situation incohérente par rapport au demandeur. C'est uniquement dans ce cas qu'il réalise son point de reprise. Grace à cet algorithme, le nombre de processus participant au point de reprise est grandement diminué.

Tous ces protocoles sont à deux phases. A chaque processus est associé un point de reprise permanent résultant de la précédente phase de création de points de reprise consistants. Pendant le déroulement du protocole de création, chaque processus maintient un point de reprise temporaire qui remplacera le permanent seulement si le protocole s'est terminé avec succès. Un processus a un rôle de coordinateur et envoie des messages aux autres processus pour synchroniser le point de reprise consistant. Sur réception de ce message, les processus créent un point de reprise temporaire et envoient un accusé de réception. Lorsque le coordinateur a reçu tous les accusés, il envoie à chacun un message de validation. Sur réception de ce message, les processus transforment leur point de reprise temporaire en permanent. Ce protocole est relativement simple mais il nécessite un mécanisme lourd de diffusion fiable ordonnée.

2.2.3 Points de reprise indépendants - Journalisation

Dans cette approche, chaque processus effectue les points de reprise sans concertation avec les autres. Lorsqu'une faute affecte un processus, tous les processus se coordonnent pour reprendre leur exécution à partir d'un ensemble de points de reprise cohérents.

De nombreux mécanismes utilisant les points de reprise indépendants ont été proposés. L'approche de [HAD 82] s'est limitée aux systèmes de bases de données centralisées. La méthode présentée dans [KIM 86] évite l'effet domino par une coordination des machines. Elle repose sur un système sous-jacent "intelligent" pour établir automatiquement des points de reprise des processus interagissants. L'exécution d'un processus est alors structurée en bloc de recouvrement [RAN 87].

Actuellement, les techniques de points de reprise indépendants reposent sur l'analyse des communications pour limiter l'effet domino. Puisque les lignes de recouvrement ne sont pas connues pendant l'exécution normale, chaque processus doit enregistrer des informations sur les canaux de communications dans un emplacement (un *journal*) qui puisse résister à la défaillance du processus. Ces informations sont utilisées dans les phases de recouvrement pour limiter le nombre de processus à reprendre. Il existe deux grandes classes d'algorithmes :

– les algorithmes de *journalisation optimiste* qui limitent le surcoût d'accès aux journaux en réduisant la quantité de données à écrire et en sauvegardant les informations de manière asynchrone.

– les algorithmes de *journalisation pessimiste* qui enregistrent les informations de manière synchrone pour éviter tout effet domino.

Les deux approches traitent généralement des processus déterministes.

Journalisation pessimiste

Cette approche a été proposée pour supprimer totalement l'effet domino [STR 85, BOR 89]. Les messages sont sauvegardés de façon synchrone [POW 83b, BOR 83, SEN 93]. Le processus récepteur reste bloqué jusqu'à ce que le message soit écrit sur le support stable. En cas de défaillance, seuls les processus fautifs sont repris à partir de leur dernier point de reprise et ils consomment directement les messages de leur journal. Si les processus échangent beaucoup des données, le coût de ce type de journalisation peut devenir énorme. De manière générale, les protocoles à base de journalisation pessimiste entraînent un recouvrement de faute rapide pour un surcoût d'exécution important.

La sauvegarde synchrone des messages est la principale difficulté des méthodes de journalisation pessimiste. Pour garantir un certain degré de tolérance aux fautes, le journal est le plus souvent répliqué sur plusieurs sites. Le problème est de garantir la cohérence des copies. Afin d'assurer la cohérence, les méthodes pessimistes utilisent le plus souvent un protocole de diffusion fiable [BIR 87].

Journalisation optimiste

L'approche optimiste vise à limiter le taux de dégradation en fonctionnement normal. Les messages sont sauvegardés de manière asynchrone [STR 85, SIS 89, JOH 90, ELN 94]. Ainsi, plusieurs messages peuvent être groupés et écrits sur support stable en une seule opération. A tout instant, des messages sont en transit (i.e., non validés). En cas de défaillance, les messages en transit peuvent devenir orphelins et entraîner une incohérence impliquant la reprise de processus valides.

La sauvegarde asynchrone des messages limite le coût de la journalisation en permettant à l'émetteur de continuer son exécution sans attendre la confirmation de la sauvegarde des messages sur support stable.

La plupart des algorithmes optimistes utilisent un protocole de recouvrement en deux phases [BHA 88, WAN 92]. Durant la première phase, des informations sur les messages échangés sont collectées, puis utilisées dans la seconde phase pour déterminer les processus à reprendre. Wang et Fuchs retardent l'écriture des messages jusqu'au prochain point de reprise. A partir des informations collectées sur les communications, ils construisent un graphe des points de reprise qui leur permet d'exclure l'ensemble des points de reprise inutiles. Ils montrent par une simulation qu'ils réduisent ainsi le surcoût de la journalisation de messages et des points de reprise. Bhargava et Lian ont une approche similaire, les processus effectuent indépendamment leurs points de reprise. La cohérence est assurée par la détection en réception des messages non valides.

Dans la phase de recouvrement, les processus sont repris à partir d'un ensemble optimal de points de reprise. Cet ensemble est obtenu par un algorithme se déroulant en deux phases.

Emplacement des journaux

Les protocoles de journalisation reposent soit sur une sauvegarde des messages reçus [JOH 89, SIS 89, JUA 91] soit une sauvegarde des messages envoyés [JOH 87, STR 85]. Dans le premier cas, les processus sauvegardent les messages reçus sur le support stable. Lors du recouvrement de défaillance, un processus repris consomme les messages contenus dans le journal. Dans le second cas, les messages sont stockés dans la mémoire volatile des processus émetteurs. Si un processus défaille, les messages nécessaires à la réexécution du processus sont disponibles dans les journaux associés aux processus émetteurs.

Alvisi et al. [ALV 93] proposent un protocole présentant les avantages des journalisations pessimiste et optimiste. Leur algorithme permet d'éliminer totalement l'effet domino tout en évitant les blocages dus aux écritures synchrones. Leur protocole est similaire à celui de Manetho [ELN 94] et reposent sur la sauvegarde des messages émis et reçus.

3 Techniques de migration

La migration est un mécanisme qui permet de modifier dynamiquement le site d'exécution (e.g., processeur ou machine) d'un processus [SMI 88]. La migration a été implémentée dès le début des années 80 dans les premiers systèmes distribués [POW 83a]. On peut distinguer trois classes de système utilisant la migration :

- les systèmes distribués à régulation de charge ;
- les systèmes permettant une exécution transparente à distance ;
- les systèmes supportant le calcul scientifique de longue durée qui intègrent migration et points de reprise.

Dans la suite de cette section, nous étudions ces différents systèmes en présentant tout d'abord les politiques de décision de la migration, la mise en œuvre de la migration de l'état du processus et les problèmes de sensibilité aux défaillances.

3.1 Politiques de décision

La décision de migrer un processus est souvent reliée à une volonté d'optimiser le fonctionnement global du système. Les politiques les plus fréquemment mises en œuvre sont les suivantes :

Optimisation de la charge des processeurs. Le but de cette politique est de répartir équitablement la charge du système sur toutes les machines du réseau. Ceci revient à mettre en œuvre un ordonanceur distribué qui aura tendance à faire migrer les processus des processeurs les plus chargés vers les processeurs les moins chargés. Pour ce faire, il est nécessaire d'implémenter un mécanisme d'acquisition des différents paramètres relatifs à la charge. Les problèmes de mise en œuvre sont du à la difficulté d'observer et maintenir d'un état global du système. Des politiques de migration reposant sur un ordonnanceur distribué ont été implémentées avec succès dans les systèmes Mosix [BAR 89, BAR 93], Charlotte [ART 89] et dans la version Norma du micro-noyau Mach [MIL 94].

Optimisation de la charge réseau et des communications. Cette politique est duale de la précédente et vise à éviter la surcharge du réseau. Pour ce faire, les processus qui communiquent fréquemment entre eux seront regroupés sur un même site ou sur un même segment de réseau. Une variante de cette stratégie peut également être utilisée pour accroître la performance d'une application composée de plusieurs processus communicants. Cette politique a été utilisée dans le cadre du système objet Isatis pour réduire le coût des appels de méthode [BAN 95].

Réquisition d'une machine par son propriétaire. Les deux stratégies précédentes reposent sur une gestion fine de l'ensemble des ressources disponibles sur le réseau pour placer les processus. Dans un environnement de stations de travail, on constate souvent que de nombreuses machines restent inutilisée pendant des periodes assez longues. Une politique de placement relativement simple consiste à envoyer les processus d'une application parallèle s'exécuter sur les stations de travail disponibles. Lorsque le propriétaire de la machine se manifeste de nouveau [1], le ou les processus "parasites" sont migrés vers d'autres machines inactives ou vers la station initiatrice de l'application. Cette technique a notamment été mise en œuvre dans le système Sprite [DOU 91] pour l'exécution à distance de compilations parallèles [ROB 87, BAA 88] et dans le système Condor [LIT 87, LIT 88, LIT 92].

Accès à une ressource spécifique. Certaines applications requièrent l'utilisation de ressources matérielles spécifiques telles qu'un dispositif d'entrées/-sorties ou des coprocesseurs pour la gestion du graphique ou le traitement du signal. L'accès à certaines ressources telles que les entrées/sorties peut être résolu au moyen d'un appel de procédure à distance (e.g., RPC). Leur utilisation peut même être banalisée si le système offre la transparence d'accès, ce qui est le cas de Mosix et de Condor dans une moindre mesure. En revanche, le RPC peut se rélever totalement inefficace si le taux d'accès à la ressource est très élevé comme dans le cas de l'utilisation d'une mémoire d'écran. La migration

1. l'activité de l'utilisateur est détectée par la frappe sur le clavier de la station de travail.

vers le site qui supporte la ressource se présente donc comme une solution alternative. A l'inverse, l'utilisation de telles ressources est une raison de non égibilité du processus en tant que candidat à la migration.

Tolérance d'une défaillance. La défaillance d'un site peut être également la cause d'une migration de processus, si le système sous-jacent intègre migration et mécanisme de création de points de reprise. À la suite d'une défaillance, la reprise du processus est effectuée à partir du point de reprise, puisque l'état courant est perdu. La réunion de ces deux fontionnalités est offerte par les systèmes Condor [LIT 92], Dawgs [CLA 92], GatoStar [FOL 95] et Stardust [CAB 96] pour permettre l'exécution d'applications scientifiques de longue durée.

3.2 Migration de l'état du processus

On décompose généralement la migration d'un processus en trois phases [PHI 93] : le gel du processus sur son site d'origine, le transfert du contexte du processus vers le site de destination et le réveil du processus. Ces trois phases peuvent être mises en œuvre séquentiellement ou de manière concurrente pour réduire le temps de migration. On peut constater que les techniques utilisées pour implémenter ces phases sont similaires ou duales des mécanismes utilisés pour la création de points de reprise et le recouvrement après défaillance.

3.2.1 Gel du processus

Avant de pouvoir migrer l'état du processus, il est nécessaire de suspendre son exécution dans un état cohérent, notamment vis à vis du système d'exploitation. Des systèmes tels que Charlotte, Mosix et Sprite interdisent la migration pendant l'exécution d'un appel système ou lorsque le processus est engagé dans une communication synchrone. La migration du processus est repoussée après la terminaison de l'appel et la réception du message.

3.2.2 Transfert de l'état

Le temps de transfert de l'état du processus sur un autre site est proportionnel à la taille du contexte [ZAY 87]. En conséquence, le transfert de l'état du processus est la phase cruciale du point de vue des performances dans la mise en œuvre d'un mécanisme de migration. Il limite de ce fait le domaine d'applicabilité du mécanisme de migration. Autrement dit, plus le temps de transfert est long, plus il est difficile d'amortir le coût de la migration.

Transfert groupé Cette stratégie est la plus simple ; elle consiste à transférer en une fois l'ensemble du contexte vers le site de destination. Cette méthode a été implémentée dans les premiers systèmes de migration tels que Charlotte, DEMOS/MP et Locus [BUT 94]. La simplicité de cette technique est obtenue au détriment des performances puisque le temps de transfert est directement proportionnel à la taille du contexte. En revanche, son avantage est que le site de destination est insensible à une défaillance du site d'origine une fois la migration achevée.

Une variante de cette stratégie a été implémentée dans le système Condor. Dans ce cas, l'ensemble du contexte est transféré dans un fichier point reprise sur un disque accessible par l'ensemble des sites du réseau via un système de gestion de fichiers distribué tel que NFS ou AFS. À partir de ce point de reprise disque, le processus peut être ensuite chargé, puis relancé sur n'importe quelle machine. Cette méthode est également reprise et étendue dans le système Stardust [CAB 96], pour reconfigurer un ensemble de processus communicants après la sauvegarde d'un point de reprise coordonné.

Afin de réduire le temps global d'arrêt du processus, des stratégies de pré-copie et de copie paresseuse ont été proposées dans le but d'anticiper le redémarrage du processus sur le site de destination. Ces stratégies reposent sur l'utilisation de la mémoire virtuelle et du mécanisme de pagination.

Pré-copie La technique de pré-copie ou copie anticipée a été introduite dans le système V [THE 85]. Le principe consiste à lancer le transfert des pages mémoire du contexte dès la décision de migration tout en retardant le gel effectif du processus. Comme le processus continue de s'exécuter, il peut re-modifier des pages déjà transférées. En conséquence, le transfert des pages est réalisé de manière itérative jusqu'au moment où le nombre de pages modifiées est inférieur à un seuil déterminé. Le processus est alors gelé et les pages restantes sont migrées avec le reste du contexte (i.e., système et processeur). On peut noter qu'une variante de cette méthode a été introduite pour optimiser la sauvegarde points de reprise (cf 2.1).

Cette méthode est particulièrement intéressante lorsque le processus possède une bonne localité et qu'il n'y a pas de phénomène de compétition entre la modification des pages et leur transfert. Dans le cas inverse, cette technique peut engendrer une surcharge système et réseau importante par le fait de transferts répètés de la même page.

Copie paresseuse La copie paresseuse repose sur un transfert immédiat du contexte système vers le site de destination. Les pages mémoire du contexte sont transférées ultérieurement lors de leur référence par le processus. Cette technique a été introduite dans le système Accent [ZAY 87]. Son bénéfice majeur est que le temps de transfert devient indépendant de la taille du contexte mémoire et la durée de blocage du processus est très courte. En revanche, la somme des durées des blocages ultérieurs provoqués par chaque référence d'une page absentes peut être supérieur à celui d'un transfert en masse par le fait de la fragmentation des transferts.

Une technique hybride entre le transfert groupé et la copie paresseuse est utilisé dans Mosix [BAR 93]. Le transfert groupé est utilisé pour transférer les pages mémoire qui ont été modifiées par le site d'origine, les autres pages sont transférées à la demande par le mécanisme de pagination (e.g., *swappeur*) réparti qui repose lui-même sur le système de fichiers distribué de Mosix.

Une variante de la copie paresseuse est également utilisée dans Sprite et consiste à paginer vers un disque le contexte mémoire ; le transfert vers le site de destination est également effectué à la demande sur référence. Cette technique est intéressante si l'un des deux site possède un disque local, car elle diminue la sensibilité aux défaillances. Dans le cas contraire, il est nécessaire d'utiliser une troisième machine serveur de fichier, ce qui double le nombre de transferts réseau.

Reprise du processus La phase de reprise du processus consiste en une recréation du processus sur le site de destination. Certains contrôles sont effectués à ce moment notamment en ce qui concerne la disponibilité de réservation des ressources. Si le système offre une fonctionnalité évoluée telle que la transparence globale, il y a également conversion ou recréation du contexte système. Nous détaillons plus précisément les mécanismes liés à la transparence dans la section 4.

3.3 Sensibilité aux défaillances

Dans un environnement de stations de travail, la probabilité d'un arrêt/-redémarrage d'une machine n'est pas négligeable [2]. Les causes de redémarrage les plus fréquentes sont dues soit à une maintenance système, à une erreur d'un utilisateur et de plus en plus rarement à une défaillance matérielle.

Dans un souci de minimiser les conséquences d'une défaillance, il est nécessaire de réduire les dépendances résiduelles, c'est à dire les informations détenues par une machine et essentielles au fonctionnement d'une autre. Par exemple, la technique de copie paresseuse proposée dans Accent entraîne une dépendance résiduelle due aux pages qui restent sur le site source. Le nommage uniforme pose également un problème si seul le site d'origine est utilisé pour conserver la localisation réelle des entités (cas du système Mosix). Enfin, dans Sprite la définition de la transparence implique une totale dépendance entre un processus migré et son site d'origine (cf 4.2.1).

Les systèmes qui intègrent à la fois points de reprise et migration tendent à éliminer complètement les dépendances résiduelles entre les sites d'origine et de destination. Par exemple dans Condor et Startdust, le point de reprise disque contient toutes les informations sur le ou les processus pour assurer la reprise l'exécution. Il n'y a aucune liaison entre les processus avant et après la migration et le recouvrement. Toutefois, ces deux systèmes sont dépendants de la disponibilité du serveur de fichiers qui mémorise les points de reprise. Une première solution à ce problème repose sur l'utilisation d'un système de fichiers tolérant les fautes tel que HA-NFS [BHI 91] ou FT-NFS [NP 96]. Une deuxième solution repose sur l'intégration d'un système de fichiers répliqué dans le système de migration. Cette dernière solution a été implémentée dans

2. Clark et McMillin ont mesuré une durée moyenne de 2,7 jour entre deux arrêts [CLA 92].

le système GatoStar.

4 Transparence et niveau d'implémentation dans le système

La transparence est la propriété qui permet de masquer l'implémentation des mécanismes de migration ou de point de reprise au processus. Ces mécanismes offrant des propriétés relatives à la qualité de service offerte par le système, il est souhaitable que leur emploi ne nécessite pas de modification des sources de l'application. Par ailleurs, le niveau de transparence offert dicte également l'étendue du domaine d'application du mécanisme. Par exemple, si la transparence d'accès aux fichiers n'est pas réalisée, seules les applications scientifiques seront concernées.

4.1 Niveau d'implémentation du mécanisme

La difficulté de mise en œuvre de la transparence est proportionelle au niveau ou le mécanisme est implanté. Si celui-ci est intégré au noyau, l'ensemble des structures systèmes sont directement accessibles et manipulables. Le système résultant est généralement efficace et des stratégies très fines peuvent être implémentées. Toutefois, la diffusion des développements est limitée pour des raisons de perénnité du noyau d'origine, de difficulté de portage ou plus simplement de confidentialité des sources dans le cas d'un noyau industriel. Cette approche est historiquement la première a avoir été utilisée dans des systèmes tels que DEMOS/MP, Charlotte, Sprite et Mosix pour la migration et Manétho, Targon [BOR 89] et DWAGS pour les points de reprise.

Si le mécanisme est implanté en dehors du noyau, dans l'espace utilisateur, il est alors nécessaire de récuperer les informations systèmes et de les forcer, lors du recouvrement ou de la migration, au moyen des appels systèmes existants. Ceci n'est pas toujours possible. Il en découle que les solutions adoptées pour la mise en œuvre des mécanismes sont souvent moins fines et moins efficaces que celles d'une implémentation noyau. En revanche, la diffusion du mécanisme est facile. La pérennité des développements est également plus grande car les interfaces des nouvelles versions de systèmes recouvrent généralement celles des précédentes générations. Cette approche est utilisée dans le système ickpt [PLA 94, PLA 95] en ce qui concerne les points de reprise et dans les systèmes Condor et GatoStar qui conjugent migration et points de reprise.

Une troisième solution est apparue avec l'introduction des systèmes à micro-noyau tels que Mach [ACC 86] et Chorus [ROZ 88]. Ils offrent une machine abstraite de bas niveau qui permet la construction de systèmes d'exploitation modulaires au moyen de serveurs, chaque service système étant implémenté par un ou plusieurs serveurs. La modularité rend possible l'ajout de nouvelle fonctions sans avoir à modifier le noyau. Par conséquent, l'approche micro-noyau est intermédiaire, en terme de portabilité, entre les deux solutions précédentes. Cette approche a été utilisée par D. Milojicic [MIL 94] pour mettre en œuvre la migration dans la version Norma de Mach et par L. Philippe [PHI 93] et

D. Conan [CON 96] au dessus de Chorus. Mach a également été utilisé dans le système FTM [MUL 94, MUL 96] pour mettre en œuvre un système d'exploitation modulaire tolérant aux fautes. Enfin, le système Stardust qui offre migration et points de reprise pour en environnement hétérogène repose à la fois sur les micro-noyaux Mach et Chorus.

4.2 Mise en œuvre de la transparence

La mise en œuvre de la transparence requiert en particulier de pouvoir accéder et nommer toute ressource du système de manière uniforme quelque soit la machine sur laquelle le processus s'exécute. Pour les ressources utilisées en mode connecté, il faut de plus assurer que le contexte d'accès reste identique après migration ou recouvrement. Par exemple, pour un fichier ce contexte est défini par le pointeur d'accès et le mode d'ouverture.

4.2.1 Implémentation dans le noyau

Les systèmes Mosix et Sprite sont représentatifs de cette approche. Ils implémentent un système de nommage uniforme des fichiers qui permet de localiser le serveur à partir du nom interne du fichier ; les requètes sont réalisées au moyen d'un RPC. Les informations d'état sur les fichiers (e.g., la structure u) sont directement accessibles par le noyau et migrent avec le processus.

Il est à noter que la notion de transparence n'est pas équivalente dans les systèmes Mosix et Sprite. Dans Mosix, une régulation globale de la charge est appliquée à tout le système ; tout les sites sont équivalents. Dans Sprite, le résultat d'une exécution à distance d'un processus doit être totalement identique à celui d'une exécution locale. En conséquence, certains appels tels que ceux relatifs au temps ou à la liste des processus sont toujours exécutés par RPC sur le site d'origine. De même, si un processus migré crée un fils, le site d'origine du fils ne sera pas positionné avec la valeur du site courant, mais avec la valeur du site d'origine du père.

4.2.2 Implémentation dans l'espace utilisateur

Dans une implémentation dans l'espace utilisateur, les informations sur les processus et ressources utilisés doivent être récupérées par l'intermédiaire des appels systèmes existants. Cependant, toutes les informations ne sont pas accessibles par ce moyen. Une solution, introduite dans le système Condor, repose sur l'interposition entre l'application et le système d'une couche qui va intercepter les informations au moment où elles sont fournies par l'application [TAN 95]. Cette approche est notamment utilisée pour mémoriser des informations sur fichiers lors de l'appel des primites **open** et **dup**. Lors de l'édition de lien, des bibliothèques spécifiques remplacent ou recouvrent les bibliothèques standard.

Dans Condor, la transparence vis à vis du système Unix est assez poussée ; l'ensemble des informations contextuelles sur les fichiers et signaux sont sauvegardées dans le un point de reprise. Le nommage uniforme des fichiers est assuré par montage d'un système de fichiers distribué tel que NFS ou AFS. Une solution complémentaire, reposant sur des RPC vers un serveur de fichies, est également proposée pour les sites ne pouvant monter directement le système de fichiers.

Une bibliothèque de redéfinition des appels est également utilisée dans le système GatoStar pour permettre un accès transparent au système de fichiers répliqués et communiquer avec les autres processus en masquant leur localisation. L'implémentation des fonctions de migration et de point de reprise est réalisée par un ensemble de processus démons implantés sur chaque machine. La communication entre les démons et les processus utilisateurs est réalisée au moyen d'IPC et de signaux.

4.2.3 Implémentation par micro-noyau

L'avantage principal des micro-noyaux est permettre l'extension des fonctionnalités du système par ajout de serveurs. Cette solution a été utilisée dans le système FTM pour implémenter la gestion d'une ressource système (e.g., fichier, écran...) au moyen d'un serveur spécifique. Cette solution permet également d'adapater la politique de gestion de la transparence à chaque type de ressource.

5 Performance des mécanismes

Ainsi que nous l'avons vu plus haut, la performance des mécanismes de migration et points de reprise est directement liée à leur niveau d'implantation dans le système. Toutefois, il faut ajouter à ce premier facteur, des influences d'ordre technologique relatives à l'architecture de la machine. Comme la sauvegarde ou la migration d'un contexte entraîne le transfert de larges blocs de mémoire, le débit physique de la mémoire, du réseau et des disques est largement significatif dans le coût des mécanismes. De ce fait, on peut penser que l'introduction des réseaux haut débit dans un proche avenir va permettre de diminuer les temps de transferts et d'augmenter ainsi la plage d'utilisation de la migration et des points de reprise. En revanche, les différences entre machines rendent complexes la comparaison des différents travaux.

5.1 Mécanismes de migration

L'efficacité réelle des mécanismes de migration est sujet à controverse. Plusieurs travaux basés sur des simulations, des analyses et des experiences pratiques tendent à démontrer que la migration de processus n'apporte pas de gain significatif dans des réseaux locaux [EAG 88, DOU 91, SHI 92]. En effet, le gain peut paraître faible par rapport à la complexité de mise en œuvre des mécanismes de migration. Ces premiers résultats, généralement considérés comme acquis, sont contredits par d'autres études [KRU 88, HB 95]. En effet [HB 95] a démontré que les modèles de simulation de [EAG 88] faisaient des hypothèses trop restrictives.

Des évaluations en environnements réels effectuées sur GatoStar et la version Norma de Mach montrent que la migration peut apporter des gains très important par rapport à une simple stratégie de placement non préemptif. Les résultats de GatoStar montrent que le gain est d'autant plus important que l'ensemble des machines est hétérogène en terme de puissance. Cependant, comme dans tous les systèmes implantant la migration au niveau utilisateur, le mécanisme reste coûteux et ne concerne que des applications scientifiques ayant de long temps de traitement. Par exemple, le temps moyen de migration sur entre deux stations de travail Sparc reliées par un réseau Ethernet 10Mb/s est de 7 secondes pour un processus d'une taille de contexte de 1150 Ko.

5.2 Mécanismes de points de reprise

Le paramètre prépondérant dans l'évaluation de la performance d'un mécanisme de point de reprise, réside dans le surcoût en temps d'exécution du programme avec sauvegarde de points de reprise par rapport à une exécution normale du même programme. En plus de la disparité des machines utilisées, l'évaluation est compliquée par l'absence d'un jeu de programmes de test reconnu.

En dépit de la diversité des implémentations existantes, il ressort de plusieurs études hétérogènes en terme de modèle de programmation (échange de messages [ELN 92, PLA 94], mémoire partagée [CAB 95]) et en terme d'implémentation (intégré au noyau [ELN 92], micro-noyau [MUL 94, CAB 95]) que les solutions reposant sur des points de reprise coordonnés permettent d'obtenir des surcoûts en temps d'exécution compris entre 1% et 10% pour des période de l'ordre de 2mn entre deux sauvegardes de point de reprise. De telles performances sont tout a fait acceptables dans un environnement de calcul scientifique et laissent à penser les solutions reposant sur des points de reprise indépendants ne sont réellement intéressantes que pour des applications aux besoins plus spécifiques.

6 Conclusion

Les points de reprise et la migration sont deux mécanismes qui ont été très largement étudiés dans les systèmes de recherche depuis l'apparition des systèmes distribués. Dans ce document, nous avons présenté une synthèse des techniques de mise en œuvre de ces mécanimes et des systèmes dans lesquels ils ont été implantés.

Au contraire de d'autres mécanismes relatifs à la distribution tel que la gestion de fichiers, la migration et les techniques de point de reprise sont restées cantonnées dans des systèmes de recherche et ont eu peu de prolongements dans les systèmes d'exploitation industriels. Une première raison de ce relatif échec réside peut-être dans le fait que le besoin de ces mécanismes est trop marginal pour susciter une intégration dans le noyau par des industriels. Une deuxième raison est que la relative inefficacité des implémentations en espace utilisateur restreint l'utilisation de ces mécanismes aux applications scientifiques de longue durée.

Au travers de projets comme SPIN [BER 95] et Exokernel [ENG 95], on peut constater une évolution actuelle des systèmes d'exploitation vers un support de l'extensibilité dans les systèmes monolithiques. L'avantage de ces systèmes est d'offrir de très bonnes performances, en permettant l'accès à des ressources de très bas niveau sans pour autant imposer la réécriture complète du système ainsi que l'imposait les micro-noyaux. On peut donc penser que de tel systèmes seront des supports idéals pour l'implémentation des mécanismes de migration et points de reprise.

Références

[ACC 86] Accetta M., Baron R., Bolosky W., Golub D., Rashid R., Tevanian A. et Young M., Mach: A new kernel foundation for Unix development , *Proc. of Usenix 1986 Summer Conference*, p. 93–112, juillet 1986.

[ALV 93] Alvisi L., Hoppe B. et Marzullo K., Nonblocking and orphan-free message logging protocols , *Proc. of 23rd International Symposium on Fault-Tolerant Computing Systems*, p. 145–154, Toulouse, France, juin 1993.

[ART 89] Artsy Y. et Finkel R., Designing a process migration facility - the Charlotte experience , *IEEE Computer*, vol. 22(9), 1989, p. 47,56.

[BAA 88] Baalbergen E.H., Design and implementation of parallel make , *Computing systems*, vol. 1, 1988, p. 135–158.

[BAN 95] Banâtre M., Belhamissi Y., Issarny V., Puaut I. et Routeau J.P., Adaptive placement of method executions within a customizable distributed object-based runtime system: Design, implementation and performance , *Proc. of 15th International Conference on Distributed Computing Systems*, p. 279–286, Vancouver, Canada, juin 1995.

[BAR 89] Barak A. et Wheeler R., Mosix : An integrated multiprocessor unix , *Proc. of the Winter 1989 USENIX Conference*, p. 101–112, San Diego, CA, février 1989.

[BAR 93] Barak A., Guday S. et Wheeler R.G., *The MOSIX Distributed Operating System*, Springer Verlag, 1993, *Lecture Notes in Computer Science*, volume 672.

[BER 95] Bershad B.N., Savage S., Pardyak P., Gün Sirer E., Fiuczynski M.E., Becker D., Chambers C. et Eggers S., Extensibility, safety and performance in the SPIN operating system , *Proc. of 14th ACM Symposium on Operating Systems Principles*, p. 267–283, 1995.

[BHA 88] Bhargava B. et Lian S.R., Independent checkpointing and concurrent rollback for recovery in distributed systems - an optimistic approach , *Proc. of 7th Symposium on Reliable Distributed Systems*, p. 3–12, Columbus (OH), octobre 1988.

[BHI 91] Bhide A., Elnozahy E.N. et Morgan S.P., A highly available network file server , *Proc. of Winter 1991 Usenix Technical Conference*, p. 199–205, Dallas (TX), 1991.

[BIR 87] BIRMAN K. P. et JOSEPH T., Reliable communication in the presence of failures , *ACM Transactions on Computer Systems*, vol. 5, n° 1, février 1987, p. 47–76.

[BOR 83] BORG A., BAUMACH J. et GLAZER S., A message system supporting fault-tolerance , *Proc. of 9th ACM Symposium on Operating Systems Principles*, p. 90–99, Bretton Woods, N.H., octobre 1983.

[BOR 89] BORG A., BLAU W., GRAETSCH W., HERRMANN F. et OBERLE W., Fault tolerance under unix , *ACM Transactions on Computer Systems*, vol. 7, n° 1, 1989, p. 1–24.

[BRI 84] BRIATICO D., CIUFFOLETTI A. et SIMONCINI. L., A distributed domino-effect free recovery algorithm , *Proc. of 4th Symposium on Reliable Distributed Systems*, p. 207–215, octobre 1984.

[BUT 94] BUTTERFIELD D.A. et POPEK G.J., Network tasking in the locus distributed unix system , *Proc. of Usenix 1984 Summer Conference*, p. 62–71, 1994.

[CAB 95] CABILLIC G., MULLER G. et PUAUT I., The performance of consistent checkpointing in distributed shared memory systems , *Proc. of 14th Symposium on Reliable Distributed Systems*, p. 96–105, Bad Neuenahr (Germany), septembre 1995.

[CAB 96] CABILLIC G. et PUAUT I., *An Overview of Stardust: an Environment for Parallel Programming on Networks of Heterogeneous Workstations*, Rapport technique n° 1006, IRISA, avril 1996.

[CHA 85] CHANDY K. M. et LAMPORT L., Distributed snapshots : Determining global states of distributed systems , *ACM Transactions on Computer Systems*, vol. 3, n° 1, février 1985, p. 63–75.

[CLA 92] CLARK H. et MCMILLIN B., DAWGS - a distributed compute server utilizing idle workstations , *Journal of Parallel and Distributed Computing*, vol. 14, février 1992, p. 175–186.

[CON 96] CONAN D., *Tolérance aux fautes par recouvrement arrière dans les systèmes informatiques répartis*, Thèse de doctorat, Université de Paris 6, septembre 1996.

[CRI 91] CRISTIAN F. et JAHANIAN F., A timestamp-based checkpointing protocol for long-lived distributed computations , *Proc. of 10th Symposium on Reliable Distributed Systems*, p. 12–20, septembre 1991.

[DOU 91] DOUGLIS F. et OUSTERHOUT J., Transparent process migration: Design alternatives and the Sprite system , *Software Practice and Experience*, vol. 1, 1991, p. 757–785.

[EAG 88] EAGER D., LAZOWSKA E. et ZAHORJAN J., The limited performance benefits of migrating active processes for load sharing , *Performance Evaluation*, vol. 12, mai 1988, p. 662–675.

[ELN 92] ELNOZAHY E.N., JOHNSON D.B. et ZWAENEPOEL W., The performance of consistent checkpointing , *Proc. of 11th Symposium on Reliable Distributed Systems*, p. 39–47, Houston (TX), octobre 1992.

[ELN 94] ELNOZAHY E.N. et ZWAENEPOEL W., On the use and implementation of message logging , *Proc. of 24th International Symposium on Fault-Tolerant Computing Systems*, p. 298–307, Austin (Tx), juin 1994.

[ENG 95] Engler D.R., Kaashoek M.F. et O'Toole J.W., Exokernel: An operating system architecture for application-level resource management , *Proc. of 14th ACM Symposium on Operating Systems Principles*, p. 251–266, 1995.

[FOL 95] Folliot B., Raverdy P. et Sens P., Plate-forme pour la répartion de charge et la tolérance aux fautes des applications parallèles en environnement réparti , *Calculateurs Parallèles*, vol. 7, Ndeg. 4, 1995.

[HAD 82] Hadzilacos V., An algorithm for minimizing roll back cost , *Proc. of ACM Symposium on PODS*, p. 93–97, 1982.

[HB 95] Harchol-Balter M. et Downey A. B., *Exploiting Lifetime Distributions for Dynamic Load Balancing*, Rapport technique n° UCB/CDS-95-887, Berkeley (CA), University of California, novembre 1995.

[ISR 89] Israel S. et Morris D., A non-intrusive checkpointing protocol , *Proc. of the Phoenix Conference on Communication and Computer*, p. 413–421, 1989.

[JOH 87] Johnson D.B. et Zwaenepoel W., Sender-based message logging , *Proc. of 17th International Symposium on Fault-Tolerant Computing Systems*, p. 97–104, Pittburg (PE), juin 1987.

[JOH 89] Johnson D.B., *Distributed System Fault Tolerant Using Message Logging and Checkpointing*, Thèse de PhD, Rice University, décembre 1989.

[JOH 90] Johnson D.B. et Zwaenepoel W., Recovery in distributed systems using optimistic message logging and checkpointing , *Journal of Algorithms*, vol. 13, n° 3, septembre 1990, p. 462–491.

[JUA 91] Juang T.T-Y et Venkatesan S., Crash recovery with little overhead , *Proc. of 13th International Conference on Distributed Computing Systems*, p. 454–461, Arlington (TX), mai 1991.

[KAA 92] Kaashoek M.F., Michiels R., Bal H.E. et Tanenbaum A.S., Transparent fault-tolerant in parallel orca programs , *Proc. of the Symposium on Experience with Distributed and Multiprocessor Systems III*, p. 297–312, mars 1992.

[KIM 86] Kim K.H., You J.H. et Abouelnaga A., A scheme for coordinated execution of independently design recoverable distributed processes , *Proc. of 16th International Symposium on Fault-Tolerant Computing Systems*, p. 130–135, Vienne, Autriche, juillet 1986.

[KOO 87] Koo R. et Toueg S., Checkpointing and rollback-recovery for distributed systems , *IEEE Transactions on Software Engineering*, vol. 13(1), janvier 1987, p. 23–31.

[KRU 88] Krueger P. et Livny M., A comparison of preemptive and non-preemptive load distributing , *Proc. of the 8th International Conference on Distributed Computig System*, p. 123–130, juin 1988.

[LEU 88] Leu P. et Bhargava B., Concurrent robust checkpointing and recovery in distributed systems , *Proc. of 4th International Conference on Data Engineering*, p. 154–163, Los Angeles (CA), février 1988.

[LI 90] Li K., Naughton J.F. et Plank J.S., Real-time, concurrent checkpoint for parallel programs , *Proc. of the 1990 Conference on the Principles and Practice of Parallel Programming*, p. 79–88, mars 1990.

[LIT 87] Litzkow M., Remote UNIX - turning idle workstations into cycle servers , *Proc. of Usenix 1987 Summer Conference*, p. 381–384, juin 1987.

[LIT 88] Litzkow M., Livny M. et Mutka M., Condor - a hunter of idle workstations , *Proc. of 8th International Conference on Distributed Computing Systems*, p. 104–111, juin 1988.

[LIT 92] Litzkow M. et Solomon M., Supporting checkpointing and process migration outside the Unix kernel , *Proc. of the Usenix Winter Conference*, p. 283–290, janvier 1992.

[MIL 94] Milojicic D.S., *Load Distribution: Implementation for the Mach Microkernel*, Vieweg Publishing, 1994.

[MUL 94] Muller G., Hue M. et Peyrouze N., Performance of consistent checkpointing in a modular operating system: Results of the FTM experiment , *Dependable Computing - EDCC1* éd. par Echtle K., Hammer D. et Powell D., p. 491–508, Springer Verlag, octobre 1994.

[MUL 96] Muller G., Banâtre M., Peyrouze N. et Rochat B., Lessons from FTM: an experiment in the design & implementation of a low cost fault tolerant system , *to appear in IEEE Transaction on Reliability*, juin 1996.

[NP 96] N. Peyrouze G. Muller, FT-NFS: an efficient fault tolerant NFS server designed for off-the-shelf workstations , *Proc. of 26th International Symposium on Fault-Tolerant Computing Systems*, Sendai (Japan), juin 1996.

[PHI 93] Philippe L., *Contribution l'étude et la réalisation d'un système d'exploitation image unique pour multicalculateur*, Thèse de doctorat, Université de Franche-Comté, 1993.

[PLA 94] Plank J.S et Li K., Performance results of ickp - a consitent checkpointer on the ipsc/860 , *Scalable High Performance Computing Conference*, Knoxwill, TN (USA), mai 1994.

[PLA 95] Plank J.S., Beck M., Kingsley G. et Li K., Libckpt: Transparent checkpointing under unix , *Proc. of 1995 Usenix Technical Conference*, 1995.

[POW 83a] Powell M.L. et Miller B.P., Process migration in DEMOS/MP , *Proc. of 9th ACM Symposium on Operating Systems Principles*, p. 110,119, 1983.

[POW 83b] Powell M.L. et Presotto D.L., A reliable broadcast communication mechanism , *Proc. of 9th ACM Symposium on Operating Systems Principles*, p. 100–109, 1983.

[RAN 75] Randell B., System structure for software fault tolerance. , *IEEE Transactions on Software Engineering*, vol. 1, n° SE-2, juin 1975, p. 220–232.

[RAN 87] Randell B., *The Evolution of Fault-Tolerant Computing*, chap. Design Fault Tolerance, p. 251–270, Springer Verlag, 1987volume 1.

[ROB 87] Roberts E. et Ellis J., *Parmake and dp: Experience with a Distributed, Parallel Implementation of make.*, Rapport technique n° CMU/SEI-87-SR 5, Carnegie Mellon University, novembre 1987.

[ROZ 88] Rozier M., Abrossimov V., Armand F., Boule I., Gien M., Guillemont M., Herrmann F., Léonard P., Langlois S. et Neuhauser W., The Chorus distributed operating system , *Computing Systems*, vol. 1, n° 4, 1988, p. 305–370.

[RUS 77] Russell D.L., Process backup in producer-consumer systems , *Proc. of ACM Symposium on Operating Systems Principles*, 1977.

[SEN 93] SENS P. et FOLLIOT B., Star: A fault tolerant systems for distributed applications , *Proc. of the 5th IEEE Symposium on Parallel and Distributed Processing*, p. 656–660, Dallas (Tx), décembre 1993.

[SHI 92] SHIVARATRI N., KRUEGER P. et SINGHAL M., Load distributing for locally distributed systems , *IEEE Computer*, décembre 1992, p. 33–44.

[SIL 92] SILVA L.M. et SILVA J.G., Global checkpointing for distributed programs , *Proc. of 11th Symposium on Reliable Distributed Systems*, p. 155–162, Houston (TX), octobre 1992.

[SIS 89] SISTLA A. P. et WELCH J. L., Efficient distributed recovery using message logging , *Proc. of the 8th Annual ACM Symposium on Principles of Distributed Computing*, août 1989.

[SMI 88] SMITH J.M., A survey of process migration mechanisms , *ACM Operating Systems REview*, vol. 22(3), 1988, p. 28–40.

[STR 85] STROM R. E. et YEMINI S., Optimistic recovery in distributed systems , *ACM Transactions on Computer Systems*, vol. 3, n° 3, août 1985, p. 204–226.

[TAM 84] TAMIR Y. et SEQUIN C., Error recovery in multicomputers using global checkpoints , *Proc. of 1984 International Conference on Parallel Processing*, p. 32–41, août 1984.

[TAN 95] TANNENBAUM T. et LITZKOW M., Checkpoint and migration of unix processes in the Condor distributed processing system , *Dr Dobb Journal*, février 1995, p. 40–48.

[THE 85] THEIMER M.M., LANTZ K.A. et CHERITON D.R., Preemptable remote execution facilities for the V-System , *Proc. of 10th ACM Symposium on Operating Systems Principles*, p. 2–12, Washington, décembre 1985.

[TON 89] TONG Z., KAIN R.Y. et TSAI W.T., A lower overhead checkpointing and rollback recovery scheme for ditributed systems , *Proc. of 8th Symposium on Reliable Distributed Systems*, p. 12–20, 1989.

[WAN 92] WANG Y.M. et FUCHS W.F., Optimistic message logging for independent checkpointing in message-passing systems , *Proc. of 11th Symposium on Reliable Distributed Systems*, p. 147–154, octobre 1992.

[XU 93] XU J. et NETZER R.H.B, Adaptive independent checkpointing for reducing rollback propagation , *Proc. of the 5th IEEE Symposium on Parallel and Distributed Processing*, p. 754–761, décembre 1993.

[ZAY 87] ZAYAS E. R., Attacking the process migration bottleneck , *Proc. of 11th ACM Symposium on Operating Systems Principles*, p. 13–24, 1987.