Rapport INRIA 1994 — Programme 1

Construction de Systèmes et d'Applications Distribués

Projet SOLIDOR

3 mai 1995

Projet SOLIDOR

Construction de Systèmes et d'Applications Distribués

Solidor est le nouveau nom pris par le projet LSP depuis avril 1994.

Localisation : *Rennes*

Mots-clés : architecture extensible (1), architecture répartie (1), micro-noyau (1), plate-forme d'expérimentation (1), programmation répartie (1), serveur fiable (1), SOLIDOR (1), système à objets (1), système extensible (1), système réparti (1), tolérance aux fautes (1).

Solidor est un projet commun Inria/CNRS (URA 227).

1 Composition de l'équipe

Responsable scientifique

Michel Banâtre, DR Inria

Secrétaire

Evelyne Livache, SAR Inria

Personnel Inria

Valérie Issarny, CR
Christine Morin, CR
Gilles Muller, CR

Ingénieurs experts Inria

Philippe Joubert, IR
Thierry Leconte, IR
Vicente Sanchez-Leighton, IR, depuis septembre 1994

Personnel Ura 227

Pascale Le Certen, maître de conférences, université de Rennes I
Isabelle Puaut, maître de conférences, Insa
Jean-Paul Routeau, IR CNRS (atelier)

Chercheurs doctorants

Manuel Billot, bourse Thomson, depuis octobre 1994
Christophe Bidan, bourse Bull, depuis octobre 1994
Ciarán Bryce, bourse Inria-Cies, jusqu'en octobre 1994
Gilbert Cabillic, bourse Inria-Intel
Mireille Hue, bourse Cifre-Bull
Anne-Marie Kermarrec, bourse Dret
Frédéric Leleu, bourse Cifre-IPS
Erwan Moysan, bourse MESR
Nadine Peyrouze, bourse Cifre-Bull, jusqu'en octobre 1994
Christophe Thivet, bourse Cifre-Bull, depuis octobre 1994

Autres personnels

Benoît Dupin, scientifique du contingent, depuis septembre 1994
Alain Gefflaut, scientifique du contingent, jusqu'en septembre 1994
Bruno Rochat, ingénieur Bull

2 Présentation du projet

Les évolutions techniques actuelles, aussi bien dans le domaine des microprocesseurs que celui des réseaux de communication haut débit, sont à l'origine des architectures distribuées extensibles dont la puissance (calcul, stockage, nombre de connexions, ...) est fonction du nombre d'éléments (machines, processeurs, ...) interconnectés.

Leurs champs d'utilisation relèvent aussi bien des applications distribuées de type client-serveur , dans lesquelles le service peut-être lui-même implanté de façon distribuée, que du traitement d'applications parallèles nécessitant une puissance de calcul élevée et traditionnellement mises en œuvre sur des machines multiprocesseurs. C'est l'étude des architectures et systèmes distribués extensibles, de la construction

de services qu'ils supportent, la valorisation des résultats obtenus *via* des applications industrielles d'envergure qui motive nos travaux actuels. Plus précisément, nos activités de recherche s'articulent autour de quatre axes principaux que nous introduisons.

Programmation d'applications distribuées : notre activité dans le domaine de la programmation d'applications distribuées s'est jusqu'à ce jour concentrée sur la programmation concurrente à objets, nos études les plus récentes dans ce domaine ont porté sur la tolérance aux fautes, la sécurité et l'accroissement du degré de parallélisme.

Nous orientons nos recherches actuelles pour répondre aux exigences de la programmation d'applications au-dessus d'architectures distribuées extensibles (e.g., sécurité-confidentialité, réutilisation).

Conception de mécanismes système adaptés aux architectures distribuées : nos travaux relatifs aux systèmes d'exploitation distribués ont porté sur deux aspects complémentaires. Nous nous sommes intéressés à la conception et la réalisation de systèmes d'exécution à objets pour applications concurrentes, au-dessus d'architectures distribuées. Un effort tout particulier a été réalisé pour traiter les problèmes de répartition de charge (processeur, mémoire). Nous avons également étudié une mise en œuvre efficace de la protection àgrain fin (e.g., au niveau de l'objet) dans les noyaux de systèmes d'exploitation.

Systèmes et architectures tolérants aux fautes : la disponibilité (i.e., dans notre cas, la continuité de service en présence de fautes) est essentielle pour les architectures extensibles.

Nos recherches à court et moyen termes dans ce domaine se font dans deux directions.

La première est relative aux systèmes d'exploitation tolérants aux fautes fondés sur un modèle client-serveur ; nous étendons actuellement les résultats acquis dans ce cadre à un environnement Unix.

La seconde direction suivie par nos recherches concerne les architectures multiprocesseurs extensibles à mémoire partagée.

Valorisation des résultats et expérimentation : notre objectif, ici est de valoriser *via* des applications industrielles significatives nos travaux de recherche dans le domaine des systèmes distribués. Les deux

principales applications retenues aujourd'hui concernent la presse télématique et la vidéo à la demande. Une telle valorisation suppose l'existance d'une plate-forme d'expérimentation crédible au vu des applications retenues. Les composants de cette plate-forme, appelée Astrolab, sont maintenant définis (ATM, PC-Pentium, Chorus), sa mise en place vient de débuter.

Ces différents aspects sont développés dans le chapitre suivant. Compte tenu du fait que ce rapport sera utilisé lors de l'évaluation du programme 1, pour chaque thème nous avons fait quelques rappels de nos travaux depuis 1991, date de la dernière évaluation.

3 Actions de recherche

3.1 Programmation d'applications distribuées

Les travaux du projet Solidor au cours de ces trois dernières années dans le domaine de la programmation distribuée ont été plus spécifiquement centrés sur l'utilisation du paradigme objet. Ceux-ci ont notamment donné lieu à la conception et à la réalisation d'un système de programmation distribué à objets, composé d'un langage concurrent à objets (§3.1.1) et de son support d'exécution distribuée, ce dernier étant décrit dans le paragraphe 3.2.1.

Le domaine des applications développées à l'aide de ce système est celui des applications parallèles.

Suite à notre expérience dans le domaine des langages de programmation à objets, nous nous intéressons à présent à l'enrichissement de cette classe de langages avec des notions fondées sur les formalismes pour la programmation parallèle. Parallèlement à cette activité,

nous orientons nos recherches dans le domaine de la programmation distribuée pour répondre aux exigences des utilisateurs des architectures distribuées extensibles.

Nous précisons nos activités dans le domaine de la programmation à objets, incluant un rappel de nos travaux de ces dernières années, puis celles relatives à la programmation distribuée extensible dans les paragraphes suivants.

3.1.1 Programmation objet

Au cours de ces trois dernières années nos travaux dans le domaine de la programmation distribuée à objets ont été centrés sur la conception et la réalisation du langage Arche. Nous poursuivons nos travaux dans ce domaine sur trois points : la tolérance aux fautes, la sécurité et l'accroissement du degré de parallélisme.

Langage concurrent à objets Arche

Participants : Valérie Issarny, Jean-Paul Routeau

Nos principales motivations dans la conception du langage Arche étaient de répondre de manière satisfaisante aux trois points suivants :

- l'intégration du parallélisme, de l'héritage et des types abstraits dans un langage impératif fortement typé ;
- la généralisation de l'appel de méthode à un groupe d'objets ;
- l'introduction d'un mécanisme de traitement d'exceptions adapté aux contextes parallèles imbriqués.

Ces motivations ont donné lieu à un langage exhibant les caractéristiques précisées ci-après.

Une classe Arche est définie par son type et une réalisation de ce type. Le type définit une interface en regroupant les signatures des opérations accessibles. Le mécanisme d'héritage est distinct de celui de sous-typage : l'héritage permet d'étendre une réalisation et le sous-typage d'exprimer la spécialisation d'un type.

Le parallélisme est introduit principalement par la notion d'objet dans Arche qui regroupe à la fois les notions traditionnelles d'objet et de processus. Un objet est créé à partir de sa classe, ce qui conduit à la création d'un processus puis à l'appel asynchrone de la méthode d'initialisation. Les divers objets du système communiquent par appels synchrones de méthodes. Le langage intègre en outre un mécanisme de synchronisation conditionnelle qui est compatible avec celui de sous-typage. Succintement, des états de synchronisation sont définis afin de pouvoir restreindre l'ensemble des opérations accessibles de la vue. Une opération est alors définie non seulement par sa signature mais aussi par un ensemble d'états de synchronisation dans lesquels elle peut terminer son exécution. Des règles sur la définition de la synchronisation

d'un sous-type permettent de concilier les facilités de sous-typage et de synchronisation.

Le langage Arche offre la possibilité de grouper dynamiquement des objets qui sont des exemplaires de classes réalisant des types ayant un super-type commun dans la hiérarchie de sous-typage. A l'exécution, l'appel d'une méthode d'un groupe d'objets donne lieu à l'exécution de la méthode en parallèle au sein des différents objets, composants du groupe.

Enfin, le langage Arche inclut un mécanisme de traitement d'exceptions qui exprime le *modèle coopération*. La conception de ce modèle a été guidée par un souci de faciliter l'écriture de programmes robustes corrects. Ce modèle est une extension du modèle *terminaison* défini dans la littérature pour des langages séquentiels.

Concernant la mise en œuvre du langage Arche, une première version de son compilateur, générant du code C, est opérationnelle depuis septembre 92. Par ailleurs, suite à nos travaux relatifs à la conception et réalisation d'un système d'exécution distribué à objets, générique (cf. § 3.2.1), une seconde version du compilateur produisant du code C++ est opérationnelle depuis Juin 94.

Les évolutions

Participants : Ciarán Bryce, Valérie Issarny, Pascale Le Certen, Erwan Moysan

Suite à la définition initiale du langage Arche, nous avons considéré son enrichissement au travers d'études portant sur : la tolérance aux fautes logicielles, la protection et le contrôle des informations et l'accroissement du degré de parallélisme.

Tolérance aux fautes logicielles : du point de vue de la tolérance aux fautes logicielles, nous avons considéré l'extension du mécanisme de traitement d'exceptions de base de Arche de manière à supporter la facilité de recouvrement arrière.

Cette étude nous a conduits à proposer une solution au contrôle de la concurrence dans un langage à objets, en présence de recouvrement arrière, qui maintient un degré de concurrence élevé. Succinctement, cette solution consiste à permettre l'exécution parallèle d'une action standard (lectrice ou rédactrice) avec une action atomique (c'est-à-dire

pouvant faire l'objet d'un recouvrement arrière) non encore terminée même si l'action standard accède un objet modifié par l'action atomique.

Protection et contrôle des informations : L'étude relative à la protection et au contrôle des informations, effectuée en collaboration avec Jean-Pierre Banâtre, repose sur deux aspects complémentaires de ce problème : la vérification de la sécurité du flux d'information dans un système parallèle, et la définition d'un modèle de contrôle d'accès pour un langage parallèle à objets [3].

Le premier travail a donné lieu à la définition d'un système de preuve pour le langage CSP fondé sur la sémantique des flux d'information entre les variables ainsi qu'à la conception d'un algorithme de vérification statique de l'absence de transgressions de flux d'information dans les programmes séquentiels. Cette proposition du contrôle de flux a ensuite été examinée dans la perspective de son intégration dans un langage concurrent à objets. L'expression de contraintes de protection dans une application permet de répondre aux besoins d'intégrité et de sécurité des logiciels mais aussi d'avoir une meilleure compréhension du comportement du programme. A cette fin, nous avons défini un modèle de contrôle d'accès pour un langage concurrent à objets.

Accroissement du degré de parallélisme : Dans un langage de programmation tel que Arche, l'accès aux différents objets s'exprime suivant le schéma classique d'appel (synchrone) de méthodes. Il s'ensuit qu'il n'est pas possible d'exprimer simplement un modèle de communication asynchrone où l'occurrence d'un événement entraîne la modification d'un ensemble d'objets en parallèle avec l'action ayant engendré l'événement.

Dans cette perspective, nous nous intéressons actuellement à l'introduction d'ensembles d'objets et d'opérations de manipulation associées dans un langage concurrent à objets. Notre proposition repose sur le formalisme GAMMA qui permet une description abstraite de programmes, sans contraintes d'ordonnancement inutiles. La seule structure de données est le multi-ensemble ; la structure de contrôle associée, l'opérateur Γ, peut être vue comme un transformateur de multi-ensembles. Le comportement de cet opérateur peut être illustré à l'aide de la métaphore de la réaction chimique : l'exécution de Γ est une succession de réactions chimiques consommant des éléments du multi-ensemble et produisant des éléments nouveaux selon certaines règles.

Notre solution repose sur l'introduction de classes particulières qui définissent des multi-ensembles, les objets exemplaires d'une telle classe étant les éléments du multi-ensemble [13].

La transformation des multi-ensembles introduits suit le formalisme GAMMA. Elle repose sur la définition de conditions de réaction, exprimées en termes de fonctions booléennes appliquées aux objets, et d'actions de transformation qui permettent de modifier les états des objets impliqués, d'insérer de nouveaux objets, ou encore d'en retirer du multi-ensemble.

3.1.2 Programmation au-dessus d'architectures distribuées extensibles

Participants : Christophe Bidan, Valérie Issarny

Parallèlement à nos activités relatives aux langages de programmation distribués à objets, nous nous intéressons à la programmation d'applications au-dessus d'une architecture distribuée extensible. Parmi les problèmes que nous avons identifiés, nous privilégions :

- l'aide à la construction d'applications correctes et à la ré-utilisation de composants logiciels ;
- le support de l'évolution dynamique et de l'hétérogénéité tant au niveau des composants logiciels des applications qu'à celui des composants de l'environnement d'exécution sous-jacent.

L'approche préconisée pour résoudre ces problèmes qui est notamment celle retenue par l'architecture CORBA ou encore par le système d'interconnexion de logiciels Polylith, repose sur une programmation par composition, également appelée constructive, qui consiste à définir une application en termes de composition de modules logiciels (éventuellement écrits dans des langages différents). Le support d'exécution associé à une programmation par composition (parfois appelé bus logiciel) gère alors les interactions entre les modules d'une application, ceux-ci pouvant éventuellement s'exécuter sur des sites différents de l'architecture. Dans ce contexte, nos études portent plus spécifiquement sur les aspects suivants :

- définition d'une relation de sous-typage, fondée sur la notion de sous-typage comportemental, qui encourage une ré-utilisation sémantiquement correcte des composants logiciels ;

- étude des mécanismes système nécessaires au sein du bus logiciel, de manière à garantir les propriétés des applications relevant de la qualité de service, ce qui inclut notamment les propriétés de sécurité-confidentialité ;
- définition d'un modèle de communication anonyme qui permet la sélection dynamique d'un composant logiciel d'une application.

3.2 Conception de mécanismes système adaptés aux architectures distribuées

Nos activités relatives aux systèmes d'exploitation distribués ont porté sur deux aspects complémentaires. Nous nous sommes d'une part intéressés à la conception et la réalisation de systèmes d'exécution pour applications concurrentes, au-dessus d'architectures distribuées. Cette activité a donné lieu à la réalisation de deux systèmes qui sont respectivement destinés à l'exécution d'applications concurrentes à objets et à celle d'applications parallèles, fondées sur un modèle de communication par variables partagées.

Pour les deux systèmes proposés, tant l'unité de parallélisme effectif que celle de protection repose sur la notion de processus (ou tâche) du système d'exploitation sous-jacent.

Il s'avère qu'un tel modèle de calcul est pénalisant du point de vue des performances si l'on désire exécuter des applications dont certains des composants doivent être isolés des autres pour des raisons de sûreté.

A titre d'exemple simple, nous pouvons considérer l'utilisation d'un code exécutable existant dont on n'est pas assuré du bon fonctionnement. Il est alors nécessaire d'isoler ce composant pour que son éventuel disfonctionnement ne corrompe pas les autres données de l'application.

Etant donné le modèle de calcul de base, ceci requiert l'utilisation d'un processus supplémentaire qui ne se justifie pas du point de vue du parallélisme. Afin de pallier cet inconvénient, nous avons défini un nouveau mécanisme de protection pour noyaux de systèmes d'exploitation. Ce mécanisme offre une protection de grain fin au niveau de noyau et permet de réaliser l'isolation des composants d'une application au sein d'un même processus.

Nous détaillons nos travaux relatifs à la conception et mise en œuvre de systèmes d'exécution et d'un nouveau mécanisme de protection dans les paragraphes suivants.

3.2.1 Système d'exécution distribué à objets, générique : le système Isatis

Participants : Michel Banâtre, Valérie Issarny, Isabelle Puaut, Jean-Paul Routeau

Nos travaux ayant trait à la conception et mise en œuvre d'un système d'exécution à objets, ont été initialement motivés par le souci de fournir un support d'exécution distribuée efficace pour le le langage Arche. Ceci a donné lieu à la conception et à la réalisation d'un premier système à objets dédié qui est implanté au-dessus du micro-noyau de système d'exploitation Mach 3.0 et qui est opérationnel depuis l'été 93 [6]. Dans ce contexte, nous avons proposé une mise en œuvre d'une mémoire virtuelle distribuée à objets, à cohérence faible. Par ailleurs, afin de recycler les ressources (mémoire et processeur) utilisées par les objets devenus inutiles, un ramasse-miettes distribué a été défini.

Outre le fait de détruire les exécutions devenues inutiles, l'originalité de notre proposition est qu'elle ne requiert pas de synchronisation forte entre les ramasse-miettes locaux implantant le ramasse-miettes distribué [16]. L'exécution efficace des applications suppose une utilisation optimale de la ressource processeur. Ceci nous a conduit à définir des politiques de placement des calculs et de migration d'objets [2].

Du fait d'une mise en œuvre au-dessus du micro-noyau de système d'exploitation Mach 3.0, le nombre d'utilisateurs potentiels du système d'exécution du langage Arche s'avère limité comparé à une implantation au-dessus du système d'exploitation Unix. Dans un souci de plus grande diffusion de nos travaux, nous avons entrepris la conception et la réalisation d'un nouveau système d'exécution distribué à objets, le système Isatis, au-dessus du système Unix. Nous avons en outre conçu le système Isatis de manière à ce qu'il ne soit pas spécifiquement destiné à l'exécution d'applications Arche et qu'il puisse être aisément enrichi avec de nouveaux mécanismes système résultant de nos travaux de recherche. Le système Isatis est opérationnel depuis Juillet 94 ; il supporte l'exécution d'applications Arche ainsi que celles écrites dans le langage C++, enrichi de facilités d'expression de la concurrence.

Le modèle d'objet du système Isatis est classique : un objet est constitué d'un état et d'un ensemble de méthodes qui sont le seul moyen de manipuler cet état ; les objets communiquent par appels de méthodes synchrones ou asynchrones. Le modèle d'exécution offert par le système Isatis est fondé sur la notion de délégué de protection qui consiste en un ensemble distribué de tâches, chaque tâche supportant l'exécution concurrente de plusieurs activités (ou processus légers).

Suivant le modèle d'exécution du système Isatis, l'exécution d'une application repose sur l'association des objets de l'application à un même délégué de protection. La communication entre objets d'une même application est réalisée de deux manières suivant que les objets appartiennent à la même tâche ou non : dans le premier cas, un simple appel de procédure est mis en œuvre et dans le second, un appel de procédure à distance est utilisé. Enfin, la communication entre objets de délégués de protection distincts repose également sur l'appel de procédure à distance.

La généricité du système Isatis découle de la définition des objets qu'il manipule en terme d'objets C++, la facilité d'héritage de C++ permettant de raffiner le comportement des différents objets. Succinctement, nous distinguons deux types d'objets dans le système Isatis :

- les objets contrôlés par le système qui assurent la gestion respective des tâches, de l'exécution concurrente d'activités au sein d'une tâche et des communications inter-tâches ;
- les objets implantant les objets de l'application.

Concernant l'exploitation de la généricité du système Isatis, nous avons examiné l'enrichissement du système avec la facilité de placement définie pour le système d'exécution d'origine du langage Arche. Ceci nous a amenés à adapter la définition de cette facilité, initialement conçue pour un système distribué à communication par mémoire partagée, à un modèle de communication par échanges de messages. Nous avons effectué une première évaluation des performances du système résultant ; celle-ci a confirmé un accroissement global des performances des applications lorsque la facilité de placement des exécutions de méthodes est préférée à une exécution systématique des méthodes sur le site de l'objet appelé.

Toujours dans un souci d'exécution efficace des applications, nous examinons actuellement un nouvel enrichissement du système Isatis afin d'exploiter la réplication des objets générée par les placements distants

des exécutions de méthodes et d'implanter un placement des objets par migration qui vise à diminuer les coûts de communication inter-objets.

3.2.2 Système d'exécution parallèle fondé sur le partage de mémoire

Participants : Gilbert Cabillic, Isabelle Puaut

Les applications parallèles (i.e., composées d'un ensemble de processus coopérants), relèvent aussi bien du domaine du calcul scientifique (e.g., simulation de l'évolution de systèmes de corps en mouvement) que de celui de la conception assistée par ordinateur (e.g., routage et simulation de circuits intégrés). Bien que les architectures cibles de ces applications soient le plus souvent des architectures massivement parallèles, les systèmes distribués, composés d'éléments de puissance croissante, s'avèrent être des candidats attrayants pour l'exécution d'applications parallèles. Les études ayant trait aux systèmes d'exécution pour applications parallèles au-dessus de réseaux de machine portent principalement sur la réalisation de *multiprocesseurs virtuels*, c'est-à-dire l'implantation de l'abstraction d'un multiprocesseur (à mémoire partagée ou distribuée) à partir de l'ensemble de machines de l'architecture sous-jacente. Afin de construire un système d'exécution parallèle fondé sur le partage de mémoire, nous avons conçu et implanté, en collaboration avec T. Priol du projet Caps, la mémoire virtuelle partagée Myoan sur la machine parallèle à mémoire distribuée Paragon.

Myoan est un mécanisme de mémoire virtuelle partagée dont la spécificité est d'offrir à la fois un protocole de cohérence forte et des protocoles de cohérence affaiblie adaptés aux accès des applications à la mémoire partagée [18].

Des protocoles de cohérence sont notamment fournis pour les applications modifiant en concurrence des données partagées indépendantes résidant sur la même page, ainsi que pour les applications manipulant des données non modifiables.

Notre objectif final est d'implanter un multiprocesseur virtuel à mémoire partagée au dessus d'un réseau de machines hétérogènes permettant de préserver l'autonomie des machines, et de reconfigurer dynamiquement l'ensemble des machines participant à l'application. Dans ce but, une adaptation de Myoan au micro-noyau Chorus s'exécutant au dessus de

la plate-forme d'expérimentation décrite dans le paragraphe 3.4.1 est en cours de développement. Le pas suivant dans la réalisation de l'objectif final est de permettre l'interopérabilité entre les services de mémoire virtuelle partagée s'exécutant respectivement sur la machine Paragon et sur la plate-forme d'expérimentation, et de permettre la reconfiguration dynamique de l'ensemble des machines accueillant les applications.

3.2.3 Mise en oeuvre efficace d'une protection de grain fin dans les noyaux de systèmes distribués

Participants : Ciarán Bryce, Gilles Muller, Christophe Thivet

Dans la plupart des systèmes d'exploitation modernes, le concept de tâche remplit deux rôles : (i) il offre un contexte d'exécution qui abstrait la machine physique, (ii) il fournit un espace de protection entre des entités séparées. L'utilisation de la tâche comme espace de protection a été démontrée avec succès notamment pour l'implémentation de sous-systèmes spécifiques (e.g., systèmes de fichiers, couches de protocoles réseaux) grâce à la technologie micro-noyau. Cependant, cette approche devient inefficace et inappropriée lorsqu'il existe un besoin d'une protection de grain fin au sein d'une même application, qui doit alors être découpée en plusieurs tâches. Une telle situation est rencontrée lorsque l'application est construite à partir de plusieurs composants logiciels qui possèdent des niveaux de confiance différents du fait de leur provenance (e.g., réutilisation de logiciels du domaine public) ou qui doivent être exécutés avec un privilège système superviseur (e.g., spécialisation de système d'exploitation).

Afin de supporter une protection de grain fin, nous avons modifié le micro-noyau Mach afin qu'une tâche puisse posséder plusieurs *domaines de protection* au sein de son espace d'adressage [3]. Un domaine de protection définit un espace de visibilité, utilisateur ou superviseur, sur un ensemble de régions mémoires. A un instant donné, une activité (*thread*) s'exécute dans un seul domaine de protection et n'a accès qu'aux régions visibles dans ce domaine. Le changement de domaine de protection s'effectue au moyen d'un appel de procédure protégé ou *PPC* (*Protected Procedure Call*). Par rapport à un RPC (Mach) entre des tâches distinctes, un PPC permet d'éviter le surcoût dû à l'ordonnancement et au changement de contexte. De plus, l'unicité de l'espace d'adressage associé à une application est préservée ce qui permet d'éviter la conver-

sion de pointeurs lors du passage de structures complexes en paramètres et notamment d'optimiser la mise en oeuvre du PPC dans certaines situations. Nos premières mesures de performances montrent qu'un PPC non optimisé peut être exécuté en 25μs sur PC-486/66Mhz, soit quatre fois plus rapidement que le RPC Mach équivalent. Lorsqu'il est possible d'optimiser la gestion du *TLB*, le temps de d'exécution du PPC peut être réduit à une valeur de 16μs.

L'intégration des domaines de protection dans Mach a été réalisée dans un souci de compatibilité binaire avec le système Mach standard. Toutes les applications existantes, y compris le serveur Unix, peuvent être exécutées sans recourir à une compilation. Dans le futur, nous pensons intégrer les domaines de protection au sein de Chorus sur la plate-forme Astrolab (§3.4.1), afin d'effectuer une spécialisation, par application exécutée, des protocoles de communications à travers le réseau ATM.

3.3 Systèmes et Architectures tolérants aux fautes

La disponibilité (i.e., dans notre cas, la continuité de service en présence de fautes) est essentielle pour les architectures extensibles. Nous avons acquis une longue expérience dans ce domaine au travers des projets Gothic, Fasst, FTM. À partir de cette expérience, nos recherches à court et moyen termes se font dans deux directions.

La première est relative aux systèmes d'exploitation tolérants aux fautes fondés sur un modèle client-serveur (§3.3.1) ; nous étendons actuellement les résultats acquis dans ce cadre (§3.3.2) à un environnement Unix. Notre objectif à moyen terme est de généraliser nos résultats aux systèmes extensibles.

La seconde direction suivie par nos recherches concerne les architectures multiprocesseurs extensibles à mémoire partagée (§3.3.3). Nous venons de proposer une solution au problème de la disponibilité fondée sur la généralisation d'un protocole de cohérence mémoire. Notre objectif actuel est d'appliquer les résultats obtenus aux architectures distribuées.

3.3.1 Système client-serveur fiable FTM : bilan

Participants : Michel Banâtre, Mireille Hue, Gilles Muller, Nadine Peyrouze, Bruno Rochat

Notre recherche dans le domaine des systèmes client-serveur fiables s'est effectuée au travers de l'activité FTM qui s'est terminée au premier trimestre 1994 [22]. Nous rappelons brièvement les caractéristiques générales de l'architecture matérielle, du système d'exploitation, ainsi que l'ensemble des réalisations effectuées au cours de cette activité.

L'objectif de l'activité FTM était de permettre la construction de systèmes tolérant les fautes, de faible coût. L'originalité de notre approche et son faible coût reposent sur l'utilisation de machines standard (stations de travail, PCs, cartes industrielles) et de la technologie *mémoire stable*. Les caractéristiques d'une architecture FTM sont les suivantes : (i) l'architecture tolère toute faute matérielle simple, (ii) l'élément de base, appelé *site stable*, permet de relancer les calculs interrompus par une défaillance à partir d'un état cohérent des données, (iii) le réseau assure la fiabilité des communications entre deux sites stables.

Un site stable est construit à partir d'un processeur principal (ex: station de travail ou carte industrielle), d'un processeur de secours et d'une mémoire transactionelle stable (STM : *Stable Transactionnal Memory*). Une STM possède les trois propriétés suivantes : cohérence, permanence et disponibilité des données en cas de défaillance du processeur de calcul. La STM est utilisée pour mémoriser des points de reprise des processus des applications ou du système. En cas de défaillance du processeur de calcul, le contenu de la STM (points de reprise) est récupéré par le processeur de secours qui peut ensuite relancer les processus interrompus. Le processeur de secours étant inactif en fonctionnement normal, il est possible de coupler deux sites stables de façon à ce que le processeur principal d'un site stable soit le processeur de secours de l'autre et réciproquement. En conséquence, tous les processeurs de la configuration sont utilisés, ce qui contribue à réduire le coût de la tolérance aux fautes.

La technologie mémoire stable que nous avons développée dans le projet Solidor au cours de ces dernières années nous a permis de proposer plusieurs mises en œ uvre de la STM reposant soit sur une solution à base de cartes matérielles spécifiques, soit sur une solution logicielle nécessitant l'utilisation d'un lien série rapide (initialement développé

dans le projet). Comme cette seconde solution offre l'avantage d'être aisément portable sur différentes machines, nous l'avons retenue pour la construction de notre machine prototype finale.

En ce qui concerne le système d'exploitation, notre motivation a été de construire un système sûr de fonctionnement qui offre la transparence de la tolérance aux fautes aux applications utilisateur. Une des originalités de notre proposition est d'intégrer la gestion fiable des ressources au sein du système. Pour ce faire, nous avons développé un modèle de construction de systèmes sûrs de fonctionnement reposant sur la technologie micro-noyau ; ce modèle définit un ensemble modulaire de *serveurs fiables* qui gèrent chacun une ressource système (e.g., segment de mémoire virtuelle, écran, fichier).

Afin d'assurer la continuité d'exécution en cas de défaillance, les processus sont relancés à partir de points de reprise globaux cohérents (*consistent checkpointing*) calculés dynamiquement. Un point de reprise global d'une application est obtenu par coopération entre les serveurs fiables ayant exécuté des requêtes pour le compte de cette application. La construction dynamique de points de reprise globaux cohérents et le modèle client-serveur fiable sont intégrés au le micro-noyau Mach 3.0. Ce modèle est applicable à tout type de ressource système.

Les performances initiales du prototype de FTM montrent que les solutions proposées dans cette étude sont utilisables dans de nombreuses applications [14].

De plus, notre mise en œuvre n'est pas dépendante de Mach et peut être portée sur d'autres micro-noyaux tels que Chorus ou des systèmes Unix offrant des mécanismes de base similaires.

Il est aussi intéressant de noter que les résultats ont été appliqués à la mise en œuvre efficace du protocole d'appel de procédure à distance [11] en environnement non fiable.

3.3.2 Conception de services fiables

Les résultats de l'activité FTM sont à la base de travaux de recherche et de développements importants, actuellement en cours, dans le projet Solidor.

Ils concernent la conception d'un service de gestion de fichiers fiable, l'optimisation d'un système transactionnel et surtout la conception et

l'implantation d'un serveur Unix fiable. Nous donnons rapidement, dans les trois paragraphes suivants, un aperçu de ces travaux.

Conception d'un service de fichiers NFS fiable

Participants : Gilles Muller, Nadine Peyrouze, Jean-Paul Routeau

Le système de gestion de fichiers est un composant essentiel d'un système d'exploitation, que celui-ci soit centralisé ou distribué, car les fichiers sont à la base de tout travail informatique. Il est fortement souhaitable que le service rendu par le système de gestion de fichiers soit *efficace* et *sûr de fonctionnement*. Dans cette étude, nous nous intéressons plus particulièrement à la fiabilisation de NFS (*Network File System*) qui est à l'heure actuelle, le système de gestion de fichiers le plus utilisé dans un environnement de stations de travail en réseau local.

Avec la technologie actuelle des ordinateurs, l'efficacité d'un système de gestion de fichiers tel que NFS est limitée par le temps d'accès aux disques et par leur débit. Une solution communément adoptée pour optimiser les performances repose sur l'utilisation de *caches de données* en mémoire centrale, ce qui permet de diminuer le nombre d'accès disques. Toutefois, en cas de défaillance du serveur, les fichiers deviennent indisponibles à l'utilisateur, ce qui se traduit par le trop fameux message NFS server xx not responding . De plus, le contenu du cache est perdu, ce qui peut compromettre la *cohérence* des fichiers.

Afin de fiabiliser NFS sans compromettre ses performances, nous proposons d'utiliser un *cache fiable* de données reposant sur une STM logicielle dérivée de celle conçue dans le cadre de l'activité FTM. Notre argument principal est que l'accès au cache fiable en STM est beaucoup plus rapide que l'accès au disque. Comme avec l'approche suivie dans FTM, le serveur NFS est construit à partir d'un site stable constitué de deux serveurs Unix reliés par un lien série rapide, chaque serveur possédant son propre disque. De ce fait, en cas de défaillance du serveur primaire, le serveur de secours possède toutes les informations pour assurer la continuité de service. Il est à noter que notre service est indépendant du système d'exploitation et de la machine, il peut donc être mis en œuvre sur une large gamme d'architectures (Suns, PCs, ...).

Un prototype de notre service NFS fiable est actuellement en cours d'expérimentation et d'évaluation.

Système transactionnel

Participants : Michel Banâtre, Mireille Hue

Comme nous l'avons mentionné précédement, l'une des originalités de nos architectures tolérantes aux fautes est d'être construites à base de mémoires transactionnelles stables. Les deux fonctionnalités de base de la mémoire transactionnelle stable sont d'assurer la non-altération des données qu'elle stocke en cas de défaillance et d'offrir la notion d'action atomique (au sens tout-ou-rien) centralisée pour accéder à ces données.

Notre objectif, ici, est d'exploiter la notion de mémoire transactionnelle stable dans le cadre des systèmes transactionnels. Plus précisement, nous voulons montrer comment implanter un système transactionnel sur une architecture tolérante aux fautes, comparable à l'architecture de FTM, en tirant profit des actions atomiques de base offertes par la mémoire transactionnelle stable (STM). Ce but de conception revient à définir un schéma de traduction des transactions utilisateur en utilisant les transactions de base d'une STM. La définition d'un tel schéma fait l'objet des nos travaux actuels. Par ailleurs, considérant une architecture tolérante aux fautes de l'activité FTM, nous examinons la cohabitation du système transactionnel avec un modèle de tolérance aux fautes différent s'appuyant également sur la mémoire transactionnelle stable, à savoir le modèle client-serveur fiable.

Étude d'un serveur Unix fiable

Participants : Philippe Joubert, Thierry Leconte, Bruno Rochat

Notre activité a ici pour objectif de proposer et réaliser une architecture matérielle et logicielle assurant la continuité de service en présence de défaillances. Les mécanismes de tolérance aux fautes doivent être transparents aux applications s'exécutant sur le système.

L'architecture matérielle proposée reprend la notion de site stable de FTM ; elle est constituée d'une paire de machines Unix standard, chaque machine étant à la fois machine principale (elle exécute des applications) et machine de secours (elle permet la reprise des applications en cas de défaillance) de l'autre machine de la paire. L'architecture logicielle permet de ne modifier ni le système d'exploitation hôte ni les applications.

La tolérance aux fautes repose sur l'utilisation conjointe d'une technique de réplication (passive ou active) des processus Unix et de réplication active asynchrone des systèmes d'exploitation des deux machines.

Pour garantir leur faisabilité technique, la plupart des options de conception retenues ont fait l'objet soit d'un maquettage, soit d'une évaluation de performance. A ce jour, l'étude de faisabilité de notre proposition se termine.

Elle devrait se poursuivre par la réalisation d'un prototype supportant une application de taille industrielle.

3.3.3 Architectures multiprocesseurs à mémoire partagée tolérantes aux fautes

Participants : Michel Banâtre, Benoît Dupin, Christine Morin, Alain Gefflaut, Anne-Marie Kermarrec

Jusqu'au début 94 l'activité relative aux multiprocesseurs à mémoire partagée s'identifiait au projet Esprit Fasst. Depuis, nous avons initialisé une nouvelle activité (Aleth), supportée par la Dret, dont l'objectif est l'étude de la disponibilité dans les multiprocesseurs extensibles. Nous rappelons brièvement les résultats du projet Fasst, puis présentons la nouvelle activité Aleth.

Architecture multiprocesseur tolérante aux fautes (Fasst)

Notre partipation au projet Esprit Fasst (Fault-tolerant Architecture with Stable Storage Technology) a pris fin en avril 94. Nous rappelons ici les principaux résultats que nous avons obtenus dans le cadre de cette activité commencée en février 91.

L'architecture Fasst est un multiprocesseur à mémoire partagée tolérant aux fautes qui permet de tolérer les défaillances des processeurs de manière transparente aux applications et aux couches supérieures du système d'exploitation. L'originalité de l'architecture est la mémoire partagée qui est une mémoire stable récupérable (RSM), sur laquelle repose la mise en œuvre matérielle d'un protocole de récupération arrière. Les points forts de notre approche sont l'utilisation de caches et de protocoles de cohérence standard. Un seul point de récupération du système est conservé dans la RSM par le protocole de récupération arrière de type

pessimiste. L'ensemble des points de récupération des processeurs doit donc toujours former un état cohérent du système.

Lorsqu'un processeur établit un point de récupération, d'autres processeurs peuvent être forcés d'en établir un simultanément du fait du partage de données, et ce pour préserver la cohérence de l'état du système mémorisé. L'enregistrement des communications entre les processeurs par la RSM sous forme de dépendances permet de minimiser le nombre de processeurs impliqués dans une opération de sauvegarde ou de restauration d'un point de récupération.

Des simulations réalisées à partir d'un ensemble d'applications parallèles appelé Splash (*Stanford Parallel Applications for Shared Memory*) ont permis de comparer les performances obtenues pour l'approche proposée par rapport à une machine standard sans mécanisme de tolérance aux fautes, ainsi que par rapport à d'autres architectures tolérantes aux fautes à mémoire partagée (Séquoia, Carer). Notre solution exhibe une faible dégradation de performance par rapport à une architecture standard et se révèle plus efficace que les autres approches citées, car elle minimise le nombre d'opérations d'établissement d'un point de récupération. Le mécanisme des dépendances rend en outre la fréquence d'établissement des points de récupération indépendante de la charge de travail de la machine.

Multiprocesseur extensible à mémoire partagée, tolérant aux fautes (Aleth)

L'objectif de nos travaux dans le cadre de l'activité Aleth, est d'étudier des solutions à la disponibilité des données dans les architectures extensibles à mémoire partagée, qui conjuguent à la fois des intérêts d'efficacité et de limitation de coût de développement.

Notre étude a abouti, dans le cadre d'une architecture extensible de type Coma (*Cache Only Memory Architecture*), à la proposition d'un protocole de cohérence étendu qui allie la gestion de la réplication des données courantes pour l'efficacité et celle des données de récupération pour assurer leur stabilité. Ces deux types de données sont rangées à des emplacements quelconques dans les mémoires standard des nœuds et sont toutes les deux gérées par le protocole de cohérence étendu [10].

L'avantage de cette approche est de tirer parti des copies existantes d'une donnée au moment de l'établissement d'un point de récupération

de façon à réduire les transferts de données entre les nœuds. En outre, le stockage des données de récupération dans les mémoire standard des nœuds autorise leur consultation en lecture aussi longtemps qu'elles ne sont pas modifiées depuis le dernier point de récupération.

Enfin, l'absence de localisation physique fixe des données de récupérations et les mécanismes de migration de données propres aux architectures Coma simplifient la reconfiguration de l'architecture après la perte d'un nœud. Une méthodologie analogue à celle employée dans le contexte de l'activité Fasst a été utilisée pour l'évaluation de notre évaluation.

Un simulateur d'architecture extensible à mémoire partagée a été développé. Il utilise la librairie de simulation à événements discrets CSIM et le noyau de simulation Spam [5] pour le traçage d'applications parallèles réelles. Les simulations ont été réalisées pour des implémentations du protocole de cohérence étendu au sein de plusieurs architectures Coma :

- un anneau unidirectionnel de même type que celui utilisé dans la machine LXSR1 avec un protocole de cohérence employant une technique d'espionnage d'anneau.
- une architecture non hiérarchique utilisant un réseau direct en grille et un protocole de cohérence à base de répertoires.

Les résultats obtenus montrent une faible dégradation des performances par rapport à une architecture standard. Ils confirment en outre l'exploitation de la réplication naturelle des données dans les architectures Coma par le protocole d'établissement de points de récupération et attestent de l'extensibilité d'une architecture utilisant le protocole de cohérence étendu que nous proposons.

Comme nous l'avons déjà mentionné, l'objectif initial de notre étude est de tolérer la faute permanente d'un des éléments de l'architecture, sachant que la défaillance d'un processeur ou d'un module mémoire entraîne l'indisponibilité du nœ ud complet le contenant. Le protocole de cohérence étendu que nous avons défini permet de garantir la stabilité des données de récupération tout en les conservant dans les mémoires standard des nœuds. Pour assurer la reprise du système après une défaillance, plusieurs problèmes demeurent au niveau du système d'exploitation :

- redémarrer le système sans le nœud défaillant;

- reprendre l'exécution des processus qui s'exécutaient sur le nœ ud défaillant sur d'autres nœ uds en essayant de ne pas doubler la charge d'un nœ ud mais plutôt de distribuer équitablement les processus.
- localiser de manière exhaustive les données ;
- gérer le problème des entrées/sorties pour éviter la perte ou la duplication de certaines d'entres elles.

A l'issue d'une étude bibliographique relative aux systèmes d'exploitation conçus pour les architectures extensibles à mémoire partagée, nous nous attachons actuellement à définir des mécanismes de reconfiguration efficaces qui s'appuient sur les fonctionnalités offertes par le protocole de cohérence étendu.

3.4 Expérimentation et valorisation

Pour justifier les travaux importants, résumés dans ce paragraphe, que nous avons entrepris dans les domaines que sont la valorisation et l'expérimentation, il est important de situer notre approche à la recherche dans le domaine des systèmes distribués, laquelle peut être abordée de deux manières. La première consiste à proposer, à partir des technologies émergeantes (réseaux haut-débit, multiprocesseurs, ...), de nouveaux mécanismes pour l'intégration de ces nouvelles ressources dans un système. Ces mécanismes devront ensuite être adaptés, si possible, aux domaines applicatifs considérés. La seconde approche, que nous qualifions d'*orientée applications*, consiste à concevoir et mettre en œuvre à partir d'une application les mécanismes adéquats pour supporter l'exécution de cette application. Cette seconde approche prédomine dans le projet Solidor et son importance va s'accentuer si nous nous référons aux actions que nous venons d'initialiser.

Les avantages de l'approche orientée applications sont évidents. D'une part, elle permet de poser les vrais problèmes qui devront être résolus par les systèmes de demain. D'autre part, sa validation passe par une phase d'expérimentation suffisament réaliste pour convaincre l'industriel de l'intérêt des résultats produits. Au regard de nos activités de recherche énoncées précédement, les domaines applicatifs que nous considérons aujourd'hui vont de l'étude des services à haute disponibilité à celle de services de consultation de données multimédia (vidéo-à-la-demande,

édition télématique d'un quotidien) en passant par un service de calcul parallèle.

Pour ce qui est de l'expérimentation, notre objectif est de disposer d'une plate-forme réaliste, qui puisse être un banc d'essai pour les applications suscitées, et suffisament ouverte pour valider nos résultats de recherche.

Les composants de cette plate-forme qui constitue l'élément de base pour la valorisation à moyen terme des travaux du projet Solidor, sont maintenant définis (ATM, PC-Pentium, Chorus) et son installation, prévue pour fin 96, vient de débuter.

3.4.1 Plate-forme d'expérimentation de systèmes distribués extensibles

Participants : Michel Banâtre, Gilles Muller, Vicente Sanchez-Leighton, Jean-Paul Routeau

Un problème essentiel dans la conception d'un système informatique est l'analyse et l'optimisation de son comportement en utilisation réelle. S'il est relativement facile d'étudier un système ou une application sur une machine unique, le problème est beaucoup plus difficile pour les systèmes distribués. En effet, beaucoup de prototypes qui se comportent de manière acceptable pour une architecture limitée à la dizaine de machines, voient leurs performances se dégrader lorsque le nombre de machines dépasse cette limite. Pour étudier le comportement des systèmes ou d'applications distribués en grandeur réelle, il est donc nécessaire de disposer dès la construction du prototype d'une plate-forme d'expérimentation de plusieurs dizaines de machines.

La construction d'une telle plate-forme nécessite aussi de prendre en compte des problèmes tels que le suivi de l'évolution technologique et la préservation des développements. D'une part, la rapidité d'évolution des calculateurs entraîne une durée de vie relativement courte, environ deux ans, des gammes des constructeurs. La présentation de résultats de recherche sur des machines obsolètes est toujours malaisée et il est difficile d'extrapoler des résultats sur des nouvelles générations de machines. Il est donc nécessaire d'intégrer l'évolution du matériel dès la conception de la plate-forme. Cette évolution doit être graduelle, car pour des raisons évidentes de coût il n'est pas envisageable de remplacer plusieurs dizaines de machines simultanément.

D'autre part, l'évolution du matériel doit être effectuée tout en préservant les logiciels développés précédemment. Jusqu'à présent, la réutilisation de logiciels s'est avérée difficile, voire impossible. La mise en place de la plate-forme, perçue comme un dénominateur commun de développements logiciels, serait une réponse à ce problème de la pérennité des développements. Ceci suppose d'intégrer l'hétérogénéité dans nos choix tant au niveau du matériel que du système d'exploitation.

Une plate-forme informatique distribuée repose sur trois éléments : un réseau, des machines et un (ou plusieurs) systèmes d'exploitation. Notre objectif est de disposer dans les deux ans à venir d'une plate-forme composée d'environ une trentaine de machines. En fonction de nos contraintes d'extensibilité et de support de l'hétérogénéité, nous avons choisi une architecture reposant sur un réseau ATM et des machines de type PC/Pentium. En ce qui concerne le système d'exploitation, nous avons choisi le micro-noyau Chorus pour les expérimentations systèmes de bas niveau et le système Chorus/Mix pour les travaux nécessitant un système compatible Unix. Les raisons principales de ce choix sont premièrement les fonctionnalités de Chorus et sa portabilité ce qui permet une évolution aisée vers les futures générations de machines.

Cette plate-forme, dont la réalisation vient de commencer dans le cadre d'un partenariat avec la société OST pour tout ce qui concerne le réseau ATM, sera disponible dans deux ans et comportera une trentaine de machines [20].

La plate-forme Astrolab sera valorisée par nos travaux de recherche internes au projet. Ils concernent notamment la réalisation d'un multiprocesseur virtuel. L'intérêt de cette plate-forme est aussi (et surtout) démontré au travers de deux applications multimédia d'envergure.

3.4.2 Applications multimédia

Participants : Michel Banâtre, Manuel Billot, Valérie Issarny, Frédéric Leleu, Isabelle Puaut

Parmi les deux applications multimédia considérées, l'une concerne la construction d'un service presse télématique . L'objet de ce service, appelé Etel (Édition TÉLématique), est de permettre la diffusion d'un quotidien en utilisant un support informatique. Le contenu et la présentation des informations fournies sont identiques à ceux délivrés par le

support papier existant. Les problèmes critiques à résoudre pour ce service concernent en outre les temps de réponse, la disponibilité, la qualité de l'information transmise et l'extensibilité (nombre d'usagers, nombre d'éditions).

La seconde application traite d'un service de vidéo à la demande. Le rôle du projet Solidor dans la mise en place de ce service est de définir et mettre en œuvre un serveur réparti à haute disponibilité pour supporter cette application. Les problèmes, non triviaux, qui devrons être résolus pour implanter ce serveur concernent la disponibilité, le stockage réparti et l'accès en lecture des données vidéo en respectant les contraintes de temps de réponse inhérentes au service considéré. L'architecture cible du serveur réparti est la plate-forme Astrolab.

L'étude de ces deux applications venant de débuter, nous ne les développerons pas plus dans ce présent rapport. Il est à noter que le déroulement de ces deux opérations se fait en étroite collaboration avec des partenaires industriels des secteurs concernés.

4 Actions industrielles

Nos activités de recherche sont fortement liées à nos actions industrielles. Nous donnons ici l'ensemble des conventions qui sous-tendent ces travaux. Pour chacune d'entre elles, nous précisons les actions de recherche concernées

- Esprit : convention FASST n°191C0560031303006
  - partenaires : August Systems, Bull UK, ETRA et Stollman, université de Newcastle upon Tyne, Trinity College (Dublin)
  - date de début : 14/02/91, date de fin : 15/04/94
  - activité de recherche concernée : §3.3.3
- Dret : convention FTM n°190C2390031303011
  - partenaires : Bull
  - date de début : 10/10/90, date de fin : 11/02/94
  - activité de recherche concernée : §3.3.1
- Bull : convention GOTHIC n°187C0330031303012
  - date de début : 09/03/87, date de fin : 01/03/96

  - activité de recherche concernée : §3.2.1, §3.3.2 (service de fichiers NFS fiable et système transactionnel)

- Dret : convention ALETH n°194C1060031303011
  - date de début : 05/05/94, date de fin : 04/11/96
  - activité de recherche concernée : §3.3.3 (multiprocesseur extensible)
- Région Bretagne : convention Plate-forme n°494C2130031303061
  - date de début : 06/06/94, date de fin : 05/06/96
  - activité de recherche concernée : §3.4.1
- IPS : convention CIFRE n°093C3000031303062
  - date de début : 01/10/93, date de fin : 30/09/96
  - activité de recherche concernée : §3.4.2
- France Telecom : convention ST2
  - partenaires : Bull
  - date de début : 1/12/94, date de fin : 31/12/95
  - activité de recherche concernée : §3.3.2 (serveur Unix fiable)
- Intel : convention n°193C2140031318012
  - date de début : 24/06/93, date de fin : 23/06/96
  - responsable Inria : Thierry Priol, projet Caps
  - activité de recherche concernée : §3.2.2

Trois conventions avec des partenaires industriels sont entrées dans leur phase administrative finale et devraient être notifiées avant la fin de cette année (94), il s'agit de vidéo à la demande (Thomson BS), Édition Télématique EtTel (TC) et plate-forme Astrolab (OST).

5 Actions nationales et internationales

Comme nous venons de le faire pour les actions industrielles, nous donnons ici la liste des conventions qui soutiennent nos actions nationales et internationales. Bien entendu cette liste n'inclut pas les contacts informels que nous avons avec différents laboratoires ou organismes de recherche.

5.1 Actions nationales

- MESR : convention Système à objets persistants
  - partenaires : Prism (Versaille), Inria-Rocquencourt, Imag (Grenoble), LSI (Montpellier)
  - date de début : 1/12/94, date de fin : 31/12/95

Une convention est en cours de notification avec France Télécom en collaboration avec C. Consel du projet Lande.

5.2 Actions internationales

- Afirst : convention Franco-Israélienne
  - partenaires : université Hébraique de Jérusalem, Bull Imag, Inria Rocquencourt
  - date de début : 03/91, date de fin : 03/95
- Esprit : convention BROADCAST n°192A264000MC006
  - partenaires : université de Newcastle upon Tyne, Inesc de Lisbonne, université Joseph Fourier Grenoble, Inria Rocquencourt, université de Twente, université de Bologne, université de Lausanne.
  - date de début : 01/09/92, date de fin : 31/08/95
- ESPRIT : convention CABERNET n°193A18500000MC005
  - partenaires : université de Newcastle upon Tyne, EPF de Lausanne, INESC de Lisbonne, université de Twente, université de Bologne, université de Cambridge, université de Vienne, université de Kaiserslautern.
  - date de début : 24/07/92, date de fin : 23/07/95

5.3 Coopération avec des projets Inria

Nous entretenons des relations avec plusieurs projets Inria appartenant aux différents programmes.

- Programme 1 :
  - projet Caps sur le thème de la mémoire virtuelle distribuée (ERDP Intel).

  - projets ADP et Sor dans le cadre de la convention ESPRIT BROADCAST et de la convention MESR Système à objets persistants .
  - projet Model dans le cadre de la convention Fasst.
- Programme 2 :
  - projet Lande sur les thèmes de la sécurité dans l'évaluation partielle pour l'optimisation de systèmes et sur le formalisme GAMMA.
- Programme 4 :
  - projet Siames sur le thème des noyaux de système pour la synthèse d'images (stage de DEA commun).
- Nous avons aussi des contacts très suivis avec le projet Guide-(Sirac) de l'université Joseph Fourier de Grenoble.

6 Diffusion des résultats

6.1 Colloques et congrès

Des membres de l'équipe ont participé à des conférences et *workshops* ; on se reportera à la bibliographie pour en avoir la liste. Certains ont également donné des séminaires.

- Towards Highly Available Scalable Shared Memory Multiprocessors : C. Morin, Séminaire, university of Michigan, USA, décembre 1993.
- Arche : A framework for Parallel Object-Oriented Programming above a Distributed Architecture : V. Issarny, CaberNet94 Workshop, Dublin, Irlande, 23-25 Mars 1994.
- Placement of Method Executions in a Distributed Object-Based Runtime System , V. Issarny, Broadcast closed workshop, Bologne, Italie, 16-19 Mai 1994.
- Programming Paradigms for Large Scale Distributed Systems , V. Issarny, Broadcast closed workshop, Bologne, Italie, 16-19 Mai 1994.
- Myoan : A Shared Virtual Memory for the Paragon , G. Cabillic, European Intel Supercomputing User's Group (USEG Meeting), Manchester, GB, 7-8 juillet 1994.

- An Implementation of a Distributed Shared Memory above the Intel Paragon , G. Cabillic, Franco-Israelian Workshop on Distributed Algorithms and Systems, Saint Malo, France, 2-4 octobre 1994.
- Design of the Myoan Shared Virtual Memory for the Paragon XP/S , G. Cabillic, PEP94, Parallel Workshop, Bergen, Norvège, 6-7 octobre 1994.
- Experience with the Design of a Distributed Object-Based Runtime System , V. Issarny, Séminaire au IBM TJ Watson Research Center, Yorktown Heights, USA, 24 Octobre 1994.
- Efficient Treatment of Failures in RPC Systems , V. Issarny, Séminaire, university of Washington, Seattle, USA, 31 Octobre 1994.

6.2 Organisation de Workshops

- Journées inter-PRC. Ces deux journées organisées à l'Irisa ont regroupé une trentaine de participants sur le thème: les systèmes à objets et leur utilisation dans les bases de données.
- Workshop Franco-Israélien. Cette 3^{me} édition du workshop Franco-Israélien s'est déroulée du 2 au 4 octobre 1994 à Saint Malo, et 25 participants venus de différentes universités et centres de recherche français et israéliens y ont présenté leurs travaux dans le domaine des systèmes distribués. A cette occasion, le Professeur A. Barak de l'université hébraique de Jérusalem a effectué un séjour d'un mois au sein du projet Solidor.

6.3 Comité de programme

C. Morin a fait partie du comité de programme de la conférence ICDCS-14 (*International Conference on Distributed Computing Systems*) qui a eu lieu en juin 1994 à Poznan.

6.4 Actions d'enseignement

M. Banâtre est responsable du cours de systèmes d'exploitation répartis (5^{e} année Insa de Rennes option informatique). Il est également respon-

sable de l'option Systèmes distribués et calculs répartis (Sycr) du DEA d'informatique de l'IFSIC dans laquelle intervient V. Issarny.

C. Morin intervient sur le thème des architectures multiprocesseurs à mémoire partagée, dans l'option Architectures Hautes Performances (Arch) du DEA d'informatique de l'IFSIC.

M. Banâtre assure un cours de système distribué au Ceram à Nice.

Des étudiants de DEA et de 5^{me} année Insa ont effectué leur stage de fin d'études au sein de l'équipe Solidor, en voici la liste : S. Billiart, M. Billot, B. Bouchet, E. Bourcy, B. Charpiot, B. Dupin, P. Leguevel, L. Legal.

7 Publications

Livres et monographies

[1] M. BANÂTRE, P. A. LEE, (Eds.), *Hardware and Software Architectures for Fault Tolerance. Experiences and Perspectives*, *LNCS*, 774, Springer-Verlag, 1994.

Thèses

[2] Y. BELHAMISSI, *Placement des calculs et des données dans un système distribué à objets*, thèse de doctorat, université de Rennes 1, juin 1994.

[3] C. BRYCE, *Étude et mise en œuvre de propriétés de sécurité dans les systèmes informatiques*, thèse de doctorat, université de Rennes 1, octobre 1994.

[4] A. GEFFLAUT, *Proposition et évaluation d'une architecture multiprocesseur extensible à mémoire partagée tolérante aux fautes*, thèse de doctorat, université de Rennes 1, décembre 1994.

Articles et chapitres de livre

[5] A. GEFFLAUT, P. JOUBERT, «SPAM : A Multiprocessor Execution Driven Simulation Kernel», *International Journal in Computer Simulation*, mars 1994.

Communications à des congrès, colloques, etc.

[6] M. BANÂTRE, Y. BELHAMISSI, V. ISSARNY, I. PUAUT, J. P. ROUTEAU, «Arche: A Framework for Parallel Object-Oriented Programming Above

a Distributed Architecture », *in : Proceedings of the Fourteenth International Conference on Distributed Computing Systems*, p. 510–517, Poznan, Pologne, juin 1994.

[7] M. Banâtre, A. Gefflaut, C. Morin, « Une proposition d'architecture extensible à mémoire partagée à haute disponibilité », *in : actes du Séminaire Journée thématique de la DRET, Temps réel et sûreté de fonctionnement dans les applications de la défense*, Paris, mars 1994.

[8] M. Banâtre, « Systèmes Distribués Ouverts: Solutions Actuelles et Perspectives », *in : Revue Technique Thomson CSF*, Acte des journées SIC 94, Campus Thomson, 1994.

[9] C. Bryce, V. Issarny, G. Muller, I. Puaut, « Towards Safe and Efficient Customization in Distributed Systems », *in : Proceedings of 6th ACM SIGOPS European Workshop "Matching Operating Systems to Applications needs"*, p. 57–61, Warden, Allemagne, septembre 1994.

[10] A. Gefflaut, C. Morin, M. Banâtre, « Tolerating Node Failures in Cache Only Memory Architectures », *in : Proceedings of Supercomputing'94*, Washington DC, novembre 1994.

[11] V. Issarny, G. Muller, I. Puaut, « Efficient Treatment of Failures in RPC Systems », *in : Proceedings of the Thirteenth Symposium on Reliable Distributed Systems*, p. 170–180, Dana Point, USA, octobre 1994.

[12] A.-M. Kermarrec, « Structuration de systèmes d'exploitation dans les architectures NUMA », *in : actes des 6me Rencontres Françaises du parallélisme*, p. 288, ENS, Lyon, France, juin 1994.

[13] E. Moysan, « Shared Classes Mutations: Paradigms for Heterogenous Concurrent Object-Oriented Programming», *in : Proceedings of the Workshop, "Models and Languages for Coordination of Parallelism and Distribution" at ECOOP'94*, Bologne, Italie, juillet 1994.

[14] G. Muller, M. Hue, N. Peyrouze, « Performance of Consistent Checkpointing in a Modular Operating System: Results of the FTM Experiment », *in : proceedings of the First European Dependable Computing Conference*, E.-H. Powells (réd.), *LNCS*, 852, Springer-Verlag, p. 491–508, Berlin, Allemagne, octobre 1994.

[15] I. Puaut, « A Distributed Garbage Collector for Active Objects », *in : Proceedings of PARLE – Parallel Architectures and Languages Europe*, p. 539–552, Athènes, Grèce, juillet 1994.

[16] I. Puaut, « A Distributed Garbage Collector for Active Objects: Implementation, Performance Measures and Extension to an Unreliable Environment », *in : Proceedings of the 1994 OOPSLA Conference*, p. 113–128, Portland, Oregon, octobre 1994.

Rapports de recherche et publications internes

[17] M. Banâtre, A. Gefflaut, C. Morin, «Tolerating Node Failures in Cache Only Memory Architectures», *rapport de recherche n° 2335*, Inria, août 1994.

[18] G. Cabillic, T. Priol, I. Puaut, «Myoan: an Implementation of the Koan Shared Virtual Memory on the Intel Paragon», *rapport de recherche n° 2258*, Irisa, avril 1994.

[19] A.-M. Kermarrec, A. Gefflaut, G. Cabillic, C. Morin, I. Puaut, «A Proposal for a Recoverable Distributed Shared Virtual Memory», *rapport de recherche*, Inria, décembre 1994.

[20] G. Muller, M. Banâtre, «Proposition d'une architecture pour l'expérimentation de systèmes distribués extensibles», *rapport de recherche*, Inria, novembre 1994.

Divers

[21] M. Banâtre, C. Morin, «Rapport de fin de contrat Fasst», février 1994.

[22] M. Banâtre, G. Muller, «Rapport de fin de contrat ftm, no 90/346 : "Étude et réalisation d'une architecture multiprocesseur tolérante aux pannes et du système opératoire associé"», février 1994.

8 Abstract

Progress in microprocessor and communication system technology is leading to the emergence of new large scale distributed architectures whose power varies proportionally to the number of processor elements. Such architectures support execution of traditional distributed applications based on the client-server paradigm as well as massively parallel applications. This framework requires solutions to well known distributed programming issues such as protection and availability without sacrificing the underlying architecture's expected processing power.

The Solidor project is currently focusing on these large scale architectures, their operating systems, programming and utilization in multimédia applications.

Table des matières