Contributions à la modélisation et à l'optimisation des systèmes de calcul à grande échelle

Cécile Germain-Renaud

19 janvier 2005

Table des matières

Chapitre 1

Introduction

Mes travaux de recherche portent sur les environnements de calcul pour les hautes performances, et plus particulièrement leur modélisation. Jusqu'en 1999, mon activité a été centrée sur la compilation et l'expérimentation des environnements parallèles [p1-p9,p11-p26][1]. La problématique du domaine a été renouvelée avec l'apparition des grilles qui visent à la mise en œuvre du partage de ressources de calcul et de stockage massivement distribuées, en nombre et en extension géographique. L'essentiel de ce document est consacré aux travaux menés dans la période récente, depuis 2000. Ces travaux se sont déroulés dans le cadre des projets de recherche de l'équipe Architectures Parallèles du Laboratoire de Recherche en Informatique (LRI), équipe dont je suis membre depuis sa création, d'abord comme étudiante en thèse, puis comme enseignant-chercheur. La plupart de mes activités de recherche dans la période 2000-2004 sont *fortement pluridisciplinaires*, d'abord à travers des projets communs avec le Laboratoire de l'Accélérateur Linéaire (LAL), puis par mon accueil dans ce laboratoire en détachement au titre de l'Institut National de Physique Nucléaire et de Physique des Particules (IN2P3).

Les architectures matérielles et logicielles, séquentielles, parallèles ou distribuées à grande échelle constituent des systèmes artificiels, mais d'une grande complexité. Leur modélisation est indispensable pour atteindre leur objectif initial, les hautes performances. Cette modélisation implique souvent l'interaction de modèles individuellement très simples, situation dont la modélisation analytique relève au minimum d'outils mathématiques non triviaux, et peut s'avérer infaisable. Le routage hot-potatoe, auquel j'ai consacré

[1] Les références de ce type se rapportent à la liste de publications, les références ordinaires à la bibliographie.

mes premiers travaux dans le cadre du projet MEGA, est un excellent exemple de cette situation. [26] le considère comme "deceptively simple" : d'une spécification très simple, il continue à défier la modélisation analytique ; des résultats significatifs ne sont obtenus qu'en ajoutant des contraintes qui permettent une modélisation algorithmique.

Une autre approche de la modélisation est plus expérimentale : plutôt que de tenter de construire le comportement global du système à partir de ses lois locales, on étudie la loi empirique des quantités d'intérêt. Cette démarche est courante pour l'analyse du trafic réseau ; en architectures de processeurs, elle a été introduite dans les années 80 par Hennessy et Patterson, avec une immense influence sur la recherche et l'industrie ; elle est moins fréquente dans l'analyse des architectures matérielles et logicielles parallèles et distribuées. Il en existe cependant plusieurs exemples largement connus : $(n_{1/2}, r_\infty)$ pour la performance des algorithmes vectorisables, le modèle LogP pour la modélisation de la communication dans les architectures fortement couplées.

Dans le domaine de la tolérance aux défaillances, le projet ROC (Recovery Oriented Computing) est fondé sur la même approche [35] :

> Recovery Oriented Computing (ROC) takes the perspective that hardware faults, software bugs, and operator errors are facts to be coped with, not problems to be solved. et We regard the computer system as an entity that cannot be understood in its entirety, and rely on empirical observations to keep it running.

Tout le contexte du Calcul Global et Pair à Pair est fondé sur cette prémisse : traiter les fautes comme un évènement normal, et non exceptionnel.

Architecture des systèmes de Calcul Global et Pair-à-Pair [p27-p29]

Le principe du calcul Global ou Pair à Pair est d'utiliser des ordinateurs distribués géographiquement à l'échelle mondiale et communiquant uniquement par Internet pour exécuter des applications informatiques de très grande taille, et repose sur un modèle d'exploitation très novateur, qui est l'utilisation des ordinateurs d'institutions ou d'individus volontaires pendant leurs périodes d'inactivité. La problématique scientifique nouvelle, et propre à ces systèmes, est la coordination du partage des ressources de résolution des problèmes de calcul ou de stockage, dans un contexte dy-

namique, hétérogène, et surtout très peu contrôlable, que ce soit de façon centralisée ou décentralisée : les ressources sont volontaires, et peuvent disparaître du système sans préavis (volatilité des ressources). Dans ce cadre :

- J'ai participé à la définition, la réalisation et l'évaluation de performances du système de calcul à grande échelle XtremWeb développé dans le groupe Clusters et Grilles du LRI.
- J'ai initié une collaboration avec le LAL pour l'expérimentation d'XtremWeb pour l'utilisation du calcul massivement distribué comme outil d'exploitation pour le projet international d'astroparticules PIERRE AUGER.

Certification de résultats [p34]

En contrepartie de l'accès à un très grand nombre de ressources de calcul et de stockage, le contrôle exercé par un système de calcul global sur ces ressources est faible. La possibilité de corruption volontaire (malveillance) ou involontaire (maladresse) de certains résultats obtenus sous ce système est ainsi beaucoup plus réelle que dans un système centralisé La problématique générale est alors la définition de méthodes d'évaluation quantitatives de l'exactitude des résultats L'originalité de notre approche, par rapport au result-checking algorithmique, est de tenter une démarche globale d'estimation statistique du taux et de l'amplitude des défauts, incluant les risques de faux positifs aussi bien que ceux de faux négatifs, sans requérir d'information *a priori* sur la distribution des défauts, ni sur le schéma d'attaque (en particulier on ne peut exclure un comportement byzantin), ni sur la sémantique de l'application.

Communications tolérantes aux fautes [p30-p32]

La problématique générale est l'exécution interruptible et asynchrone d'applications parallèles sur un environnement volatil, où les fautes fatales sont donc très fréquentes. J'ai initié un programme de recherche dans ce domaine, en collaboration avec l'équipe Parallélisme du LRI, et par le recrutement d'un post-doctorant. Cette collaboration a abouti à :

- La définition d'un algorithme de type enregistrement pessimiste (pessimistic logging). Schématiquement, cet algorithme enregistre une trace de l'exécution ainsi que les informations nécessaires pour rejouer les processus défaillants, et eux seulement.

- La définition et la réalisation d'un environnement de passage de messages adapté à la volatilité et réalisant une implémentation complètement fonctionnelle du standard MPI. Le modèle de communication sous-jacent est l'espace de tuples : un intermédiaire, la Mémoire de Canal, découple émetteur et récepteur, et sauvegarde les messages en transit pour une future ré-exécution en cas de défaillance.

Perspectives

Mes travaux récents et mes perspectives de recherche concernent les modèles d'exploitation des grilles, avec application à l'imagerie médicale [p10,p33,p35-p37]. Le modèle le plus usuel d'exploitation des grilles de calcul reste celui du centre de calcul, essentiellement celui du traitement par lots.

Ce modèle d'exploitation exclut à priori les applications qui requièrent l'interactivité, en particulier toutes celles qui relèvent de la *supervision du calcul* (computational steering), par exemple mais non exclusivement, à partir de visualisation. L'imagerie médicale est un exemple particulièrement intéressant de cette situation. L'accès interactif à une grille favoriserait le transfert vers la pratique clinique des résultats de travaux de recherche en imagerie médicale qui reposent sur le calcul hautes performances.

Permettre l'accès interactif aux grilles pose un ensemble de questions technologiques et scientifiques, qui sont explicitées dans le dernier chapitre de ce document. Nous souhaitons particulièrement cibler deux aspects.

Passage à l'échelle du scheduling . Il ne s'agit pas tant de cibler un très grand nombre de processeurs, que de tenir compte des conditions réelles d'exécution sur une grille, qui impliquent à la fois une information lacunaire et la nécessité de médiatiser tous les processus de gestion à travers une pile de protocoles.

Un observatoire de grille . Les quelques grilles actuellement en production peuvent devenir des générateurs de données sur leur propre fonctionnement. Rassembler et analyser ces données est encore une fois un problème de passage à l'échelle : de nombreux travaux ont permis de caractériser le comportement, par exemple en disponibilité ou en charge des processeurs, pour des grappes ou des machines isolées reliées par un WAN. Corréler ces résultats au niveau d'une grille, ou quantifier des informations sur la localité des références constitue un domaine de recherche encore très peu exploré.

Chapitre 2

Travaux antérieurs

Jusqu'en 99, mes travaux de recherche ont porté sur le parallélisme pour les hautes performances. La problématique du domaine a considérablement évolué au cours de cette période. Le rôle de la conception de matériel dédié a diminué [p19]. Les composants critiques pour les hautes performances sont devenus les composants logiciels, et en particulier, le logiciel de base, compilateurs et exécutifs. La thématique de mes travaux a naturellement suivi cette évolution : la première partie concerne les architectures parallèles et la seconde la compilation parallèle

2.1 Architectures parallèles

Le projet MEGA (dirigé par J-P Sansonnet et D. Etiemble) a étudié le parallélisme ultra-massif. Dans le contexte général des architectures cellulaires, l'objectif était de démontrer la faisabilité d'une architecture MIMD à mémoire distribuée extensible jusqu'à plusieurs centaines de milliers de processeurs. Les travaux dans le prolongement de ma thèse [p3] ont défini un circuit de routage hot-potatoe randomisé sous des contraintes de surface drastiques [p4,p11,p12], obtenu une preuve formelle de terminaison pour une classe de ces routages [p4], enfin étudié son interface avec les environnements de programmation [p13] à une époque où les environnements classiques (PVM, MPI) n'existaient pas.

Le coût de communication dans une architecture massivement parallèle MIMD est en partie dû à l'asynchronisme réseau/processeurs, qui impose la mise en œuvre d'un protocole d'interface toujours coûteux. Cette analyse est

l'origine du projet PTAH (dirigé par F. Cappello) : une architecture statique à réseau reconfigurable, dans laquelle le réseau apparaît comme une unité fonctionnelle banalisée, et l'ensemble des nœuds de calcul est très fortement synchronisé [p14 ,p16].

La validité de cette approche repose sur trois hypothèses : la possibilité de construire un réseau à très haut débit [p6, p14, p18], l'existence d'une statiticité suffisante dans les applications [p7, p15, p20] , enfin la faisabilité de la compilation pour le modèle d'exécution spécifique de PTAH, que nous avons appelé modèle d'exécution statique. Certaines de ces études ont été conduites par D. Gautier de Lahaut, dont j'ai encadré la thèse.

2.2 Compilation parallèle

Les langages data-parallèles proposent une solution alternative à la programmation concurrente, qui est attractive en simplicité sémantique, et aussi du point de vue pratique en coût de développement et de maintenance. Dans ce domaine, l'analyse des communications et la génération du code de communication sont reconnues comme des questions difficiles et importantes [p22].

Outre leur intérêt intrinsèque, les langages data-parallèles, qui explicitent la synchronisation à grain fin, étaient potentiellement bien adaptés à une architecture synchrone comme PTAH. En collaboration avec F. Delaplace, nous avons défini le mécanisme de traduction des références parallèles vers des fonctions de communications, sous les contraintes d'une communication complètement compilable, en utilisant les techniques d'analyse polyédrique [p5, p17].

Les méthodes développées pour le modèle d'exécution statique se sont avérées transposables au contexte plus général de la compilation data-parallèle des applications régulières pour les architectures à mémoire distribuée, pour réaliser une optimisation cruciale pour les performances, la vectorisation des communications. Nous avons défini une méthode générale de vectorisation des communication, qui permet de traiter toutes les références affines. Cette méthode peut être prouvée optimale dans le cas des distributions cycliques [p21,p24].

J'ai également effectué des contributions dans le domaine de la compilation des applications irrégulières (gather-scatter, indirections), inspirées, soit par les solutions définies pour le modèle d'exécution statique (équivalence des accès par indirection et du tri [p23]), soit par des analogies architecturales

(optimisation du schéma inspecteur-exécuteur [p9, p26]).

Enfin, j'ai recherché la collaboration avec les utilisateurs du calcul hautes performance, qui permet de tester le réalisme des abstractions nécessaires aux travaux précédents. Sur des applications en décomposition de domaines, nous avons montré que le parallélisme de données était compétitif avec le passage de messages, mais demandait un reverse engineering des compilateurs commerciaux pour adapter la programmation [p25] ; nous avons également étudié le couplage de composants logiciels séquentiels et parallèles à travers Java [p8].

Chapitre 3

Grilles : Contextes

La prolifération des systèmes déployés, travaux expérimentaux et produits industriels gridifiés est évidente. Il serait souhaitable de voir émerger une typologie efficace, analogue à la double dichotomie MIMD/SIMD et mémoire partagée/mémoire distribuée pour les architectures parallèles, à partir de laquelle des raffinement architecturaux (UMA/NUMA par exemple) et des concepts de niveau plus élevé (parallélisme de contrôle ou de données) ont pu se bâtir. Ce n'est clairement pas encore possible : dans un travail récent [124], un des meilleurs spécialistes du domaine ne peut définir qu'une typologie descriptive des grilles à partir d'exemples déployés.

Actuellement, seule la définition de la spécificité des grilles , par rapport aux grappes et aux architectures parallèles monolithiques, est relativement claire au niveau fonctionnel, en tant que services nouveaux de globalisation et de mutualisation qu'elles doivent fournir. An niveau architectural, considéré comme celui de l'organisation interne, une distinction a émergé entre deux types de systèmes, mais la terminologie elle-même n'est pas fixée : dans le domaine institutionnel (par exemple le titre de l'Action concertée incitative GRID), pour tous les produits commerciaux et pour certains travaux scientifiques [124], le terme de grille correspond aux deux types d'architectures ; pour d'autres [93, 96], la terminologie correcte doit distinguer Grid et systèmes Pair à Pair.

3.1 Fonctionnalités

En 1998, Carl Kesselman et Ian Foster proposent une définition des Grids [94] :

> A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.

Cette définition se situe dans le prolongement des travaux sur le MetaComputing [177], en particulier l'expérience Iway [59]. Elle met donc l'accent, d'une part sur le niveau des ressources auxquelles le système permet d'accéder, d'autre part sur des exigences de qualité de service, par analogie avec la fourniture de fluides.

Cette définition correspond aux grilles au sens large, purement fonctionnel. S'agit-il d'une problématique radicalement nouvelle ? A un certain niveau de généralité, la réponse est non. L'utilisation banalisée de ressources informatiques, qui implique un accès sécurisé et transparent, une utilisation efficace et une gestion distribuée des ressources pour permettre un certain degré d'anonymisation (ou de virtualisation) est un objectif aussi ancien que l'apparition des réseaux informatiques.

3.1.1 Eléments de mise en perspective

En 1969, avec la naissance d'Arpanet, les enjeux essentiels étaient posés (document en annexe) : d'une part, la vision de l'accès banalisé, à la demande, aux ressources informatiques ; d'autre part, un résumé des principales questions : champ applicatif, l'imagerie, et ce qui en découle, l'association entre données et calcul sur la grille ; l'esquisse d'une architecture 3-tier avec un client qui soumet une requête, un broker qui orchestre les traitements et un serveur auquel sont délégués les calculs spécialisés ; cette architecture de calcul est donc d'une part très hétérogène, d'autre part dynamique, la machine C étant adjointe seulement si nécessaire ; la notion de qualité de service utilisateur (qualité de la photo) est importante ; enfin le caractère ouvert et non spécialisé, programmable, de l'architecture est affirmé.

La démarche d'agrégation de ressources indépendantes pour l'obtention de puissance de traitement, donc l'approche distribuée, est restée très importante dans la motivation de la définition des protocoles réseau :

> The objectives of FTP are 1) to promote sharing of files (computer

> programs and/or data), 2) to encourage indirect or implicit (via programs) use of remote computers, 3) to shield a user from variations in file storage systems among Hosts, and 4) to transfer data reliably and efficiently [154]
>
> It is envisioned that the protocol may also be used for (..) process-process communication (distributed computation) [153].

Dans la période suivante (85-95), c'est au contraire l'approche couplage fort, sous la forme des architectures parallèles monolithiques, puis des clusters, qui a dominé la recherche de puissance de traitement. La première explication est évidemment les contraintes des conditions matérielles, plus précisément l'écart entre la faiblesse des performances des réseaux locaux ou à longue distance, et l'évolution exponentielle des performances des processeurs.

Cette tendance s'est inversée : une étude récente évalue la progression du débit d'institutions de recherche et d'enseignement depuis les Etats-Unis vers le reste du monde, sur la base du RTT et le taux de perte [54] ; pour la plupart des régions développées, le taux d'amélioration est 80% par an, soit une multiplication par 10 en 4 ans et par 100 en 8 ans. La comparaison de ce chiffre avec l'évolution des ressources de traitement explique fondamentalement l'explosion des grilles.

- Processeurs : la densité d'intégration double tous les 18 mois (loi de Moore) ; la performance vraie est plus difficile à évaluer, par manque de benchmarks cohérents sur une longue période. En se basant sur la suite SpecInt95, on obtient un taux de progression des performances de 50% par an.
- Systèmes hautes performances déployables : au sens de machines disponibles commercialement, par opposition aux architectures ultimes, à exemplaire unique qui occupent les premiers rangs du Top500. Pour évaluer l'évolution de ce type d'architecture, nous avons considéré la centième machine du Top500. Le taux de progression est de l'ordre de 80% par an.

L'évolution de la puissance de communication, au moins en débit, est donc du même ordre que celle des architectures hautes performances, et dépasse fortement celle des processeurs. A l'échelle des grilles, le problème classique de l'aggravation du déséquilibre communication-calcul n'est pas vérifié.

Le premier environnement de grille offrant un niveau d'expressivité nettement supérieur à celui des protocoles réseau est sans doute PVM [184, 98].

Exploité actuellement surtout comme une librairie de passage de messages, plus rudimentaire que MPI, PVM est à l'origine un authentique environnement de grille, comme l'indique en partie son nom, Parallel Virtual Machine. Une API unifiée, sous forme de fonctions C, regroupe en effet les fonctionnalités suivantes : agrégation dynamique et de gestion de ressources ; masquage de l'hétérogénéité des architectures connectées, à la fois au niveau micro (processeur, système d'exploitation, représentation des données), et au niveau macro (stations de travail communiquant par TCP/IP sur un LAN ou un WAN, machines parallèles à réseau spécifique, et combinaison des deux) ; gestion des tâches, lancement, surveillance, terminaison, redirection des entrées-sorties ; enfin protocoles de communication au sens strict. Le modèle de calcul exploitable sur PVM est fondamentalement distribué (MPMD), contrairement à MPI-1 (SPMD) ; en revanche, contrairement aux protocoles de communication évoqués plus haut, la tolérance aux fautes est absente ; en ce sens, une machine construite sous PVM est effectivement parallèle, au sens de fortement couplée logiquement, même si le substrat peut être un réseau à longue distance.

Les limitations de PVM en tant qu'environnement de grille ne sont pas essentiellement liées à un modèle de communication : le système a pu évoluer vers un modèle "pluggable", à la Nexus, avec reconnaissance dynamique de protocole [98]. Il se situe plutôt au niveau de la fonctionnalité fournie. Les protocoles des années 80 voient les besoins de l'utilisateur d'une grille satisfaits par Telnet et FTP : partager, c'est avoir un terminal sur une machine distante et transférer des fichiers, de façon fiable malgré une infrastructure potentiellement fautive. PVM construit une architecture à l'intérieur d'un programme distribué ; partager, c'est disposer d'une fraction de la puissance de calcul de machines distantes pour la durée de vie d'un programme parallèle ; au moins dans la conception initiale, ces machines sont parallèles.

3.1.2 Caractéristiques nouvelles

Extensibilité

Il s'agit de virtualiser, au sens de rendre transparents, les moyens d'accès, les ressources de calcul et de stockage, à la même échelle que les protocoles réseau ont virtualisé des ressources matérielles qu'il est exclu actuellement de décrire dans leur totalité. Le problème général induit est d'offrir une représentation suffisamment cohérente et à jour d'informations beau-

coup plus complexes que des adresses IP. L'échelle visée implique d'une part une approche au moins partiellement distribuée, d'autre part l'acceptation d'approximations dans la représentation, à la différence des grappes. La volonté de définir des protocoles généralistes, ou au moins des infrastructures généralistes basées sur des protocoles standard, pour supporter cette représentation [96, 186, 5, 17], qui est commune aux recherches et à des développements industriels, n'a pour l'instant pas débouché.

Egalisation des données et du calcul

A la différence du MetaComputing, les Grids actuelles accordent une place à l'accès aux données au moins égale à celle du calcul, au niveau de systèmes de fichiers globalisés, au niveau de l'infrastructure de stockage (virtualisation des disques [81, 15]), ou comme mécanisme fondamental (systèmes Pair à pair de troisième génération, voir ci-dessous). Une conséquence est l'influence très grande de technologies orientées vers l'exploitation économique du Web sur les architectures, l'exemple le plus connu étant la définition de l'Open Grid Service Architecture (OGSA) [97] par extension des Web Services.

Deux caractéristiques complètement nouvelles sont apparues.

Le partage avec d'autres types d'utilisation des ressources

La nécessité de ce partage a été pointée dans []. Quelle motivation à long terme (autre que de réaliser des démonstrations) pour échanger des ressources entre organisations ou individus distincts ? Si la charge de travail ou de stockage est strictement constante, il n'y en a aucune : chacun ne peut que dimensionner son système avec le meilleur rapport coût-performance. L'intérêt des grilles se situe dans la variation de ces charges dans le temps sur les différents sites, qui permet d'optimiser leur répartition. La charge résiduelle sur chaque site n'étant pas nulle, et éventuellement dominante, le système de grille doit coexister avec les infrastructures locales, soit comme une surcouche (Globus), soit en admettant la préemption brutale par un utilisateur légitime (systèmes pair à pair). Ce point définit peut-être les environnements de grilles par rapport à des environnements distribués à grande échelle dédiés, par exemple Akamai [130] pour la distribution de contenus Web, et certainement par rapport aux environnements d'exploitation des clusters [204, 12], quelle que soit par ailleurs l'extension effective de la grille.

	Grilles	CG et P2P
Ressources agrégées	Structurées	Instructurées
Connexion	Haute Performance parfois dédiée	Variable éventuellement faible
Mode de partage	Institutionnel	Vol
Fautes	Classiques	Très fréquentes

TAB. 3.1 – Caractéristiques des grilles et des systèmes CG et P2P

Le franchissement de domaines d'administration distincts

Outre les questions de comptabilité avec les instruments de gestion de charge, la sécurité et la confidentialité, pour les utilisateurs grille et pour les utilisateurs non-grille doivent être assurées sans compromettre la transparence des accès, également en se fondant sur les dispositifs locaux. Une grille fournit avant tout des services de découverte et d'allocation des ressources, avec tout ce qui l'accompagne (sécurité d'une part, surveillance des ressources d'autre part).

A cette architecture sont associés des modèles de représentation des ressources, qui peuvent être très généraux, comme les paires clés-données des tables de hachage distribuées (DHT) [152, 80, 158], de type langage de description [186] ou encore objet [102, 125, 60, 7].

3.1.3 Grilles et Calcul Global et Pair-à-Pair

Deux approches du partage de ressources à grande échelle à travers des domaines administratifs distincts se sont développées à peu près simultanément : les grilles au sens strict, dont le développement le plus connu est Globus, et les systèmes de calcul global et Pair à Pair (CG-P2P dans la suite). La définition des premiers est relativement stable, celles des seconds plus controversée.

Les différences fondamentales entre les deux types de systèmes, résumées dans la table 3.1 ont pour origine le mode de partage. En 2000, la définition des grilles a été raffinée, en l'orientant vers des aspects de gestion technologique et institutionnelle [96] : la spécificité des Grids réside dans les propriétés de "coordinated resource sharing and problem solving in dynamic, multi- institutional virtual organizations." Les Grids mutualisent donc les ressources d'institutions (de recherche, économiques...) par accord mutuel ; il y a contrat sur "what to share, who is allowed to share" [95], et support du système par

les institutions participantes.

Les systèmes CG-P2P sont originaires des environnements dits de vol de cycles [91] d'une part, de partage de fichiers d'autre part (vol de stockage) ; le terme vol, traditionnel (probablement par analogie avec les mécanismes de DMA) mais inadéquat, devrait être remplacé par recyclage ; il s'agit d'utiliser les ressources normalement dévolues à un certain utilisateur lorsqu'il n'en a pas l'usage (machine en économiseur d'écran pour dénoter la non-utilisation interactive, machine de grappe non chargée, espace disque disponible). Dans ces environnements, le contrat est réduit à sa plus simple expression du côté du donneur, qui peut quitter le système à tout instant, sans notification préalable ; l'environnement doit adhérer à cette contrainte en arrêtant instantanément l'influence de son emprunt.

Une première conséquence est la nature et l'organisation des ressources agrégées. Les grilles fédèrent des ressources de haut niveau, pas nécessairement par les machines de base - EGEE/DataGrid par exemple regroupe des fermes de PC - mais par l'environnement logiciel (systèmes de batch, protection) et matériel, en particulier du point de vue de la mémoire, du stockage et du réseau. Les systèmes CG-P2P regroupent des machines individuelles, très majoritairement des PC, avec un environnement matériel de qualité médiocre (PC domestiques, machines de salles d'enseignement, machines de bureau), une connexion médiocre et éventuellement intermittente (par exemple fournie par un ISP) et un environnement logiciel individuel.

Les grilles au sens strict, comme l'a reconnu I. Foster [93], ont donc des exigences limitées du point de vue de l'extensibilité, en taille de la communauté. Par exemple, EGEE regroupe moins d'une centaine de sites, et cible quelques milliers d'utilisateurs. L'extensibilité en performances de traitement est une autre affaire : l'application phare d'EGEE, le traitement des données du futur LHC, impliquent le traitement de PétaOctets de données chaque année. Mais, du point de vue des services internes de la grille, en particulier localisation des ressources et mise en relation, des solutions modérément extensibles sont acceptables. Typiquement, Globus 2 a utilisé LDAP pour construire ces services (MDS, MetaComputing Directory Service) sous la forme d'une base de données distribuée dont les temps d'accès n'étaient pas très bons.

Les systèmes CGP2P ont évidemment des exigences d'extensibilité plus importantes, les ressources étant prises en compte individuellement.

Une deuxième conséquence est le type de tolérance aux pannes visé.

> Grid Computing addresses infrastructure, but not yet failure, whereas P2P addresses failure but not yet infrastructure. [93]

Il s'agit d'une part d'un aspect qualitatif : les organisations fédérées par une grille ont une politique de gestion des fautes que le système de grille se borne à intégrer. Il s'agit surtout d'un aspect quantitatif. Dans un système CG-P2P, des événement parfaitement normaux, comme la déconnexion (systèmes de partage de fichiers, dans certains cas calcul global) ou la reprise de l'utilisation normale d'une machine (calcul global) apparaissent comme des fautes fatales : la ressource devient brutalement inutilisable.

Nous choisirons ces deux contraintes, nature individuelle des ressources et tolérance aux fautes très fréquentes, comme définissant à la fois les systèmes de Calcul Global et Pair- à-Pair. Cette définition présente l'avantage d'être indépendante de l'architecture du système, pour laquelle, comme le souligne la citation précédente, aucune convergence n'a encore eu lieu.

3.2 Systèmes de Calcul Global et Pair à Pair

3.2.1 Systèmes de Calcul Global

Leur objectif premier est de distribuer à grande échelle du calcul, l'exemple le plus connu étant SETI@home [123]. Beaucoup de projets, académiques (Javelin [144], Bayanihan [167], Atlas [11]), ou industriels dans ce domaine ne répondent strictement qu'au premier critère, intégration directe de postes de travail, et ne prennent en compte les fautes que comme des évènements relativement exceptionnels. Ces architectures, qu'on peut par exemple appeler Calcul Volontaire, visent à intégrer des machines faiblement couplées, pour répartir du calcul séquentiel [123], maître-esclave [144, 106] ou parallèle [167] ; leur objectif a plutôt été d'étudier une ou plusieurs des questions classiques des systèmes distribués renouvelées par le calcul à grande échelle : répartition de charge, en particulier par vol de travail, certification de résultats. Les codes distribuables sont le plus souvent limités au langage Java, et embarqués dans des applets : ceci simplifie beaucoup les problèmes de sécurité, et d'un point de vue pratique le recrutement de volontaires - il suffit d'activer un lien dans une page Web pour collaborer - mais limite gravement le réalisme de l'architecture (pour une discussion détaillée de cet aspect, voir la thèse de Gilles Fedak [89])

L'un des systèmes de Calcul Global les plus anciens, Condor [135], répond

au contraire aux deux critères, mais à l'échelle d'un réseau local. L'utilisateur légitime d'une machine définit des conditions d'utilisation, incluant un délai de tolérance pour la disparition du travail extérieur, délai qui peut être nul. Le système repose, d'une part sur un mécanisme de reprise en migration sur sauvegarde, d'autre part sur la localisation des actions à sémantique locale (Entrées/Sorties) sur la machine cliente via un mécanisme d'interposition (RPC et processus fantôme). La localisation des ressources suit un modèle centralisé, mais ouvert. Les demandes et offres de ressources sont décrites par un langage (ClassAd [186]), qui permet d'intégrer dynamiquement des caractéristiques quelconques ; elles sont transmises périodiquement par des agents à une instance centralisée, qui réalise l'appariement (*matchmaker*). Sur notification de leur compatibilité, les agents compatibles entrent en lien direct, et peuvent approfondir le dialogue suivant des protocoles que leur ClassAd respectifs ont indiqué être communs. Condor a donc des caractéristiques intermédiaires, les unes typiques d'un système CG-P2P, par la gestion de la volatilité, d'autres véritablement P2P, par la symétrie entre ressources et demandes traitées par appariement, les autres enfin associées à un réseau local, par la reprise sur sauvegarde, qui demande un débit réseau significatif [155], et aussi par l'absence de modèle de sécurité malgré la symétrie client/worker, qui n'est pas compatible avec un accès incontrôlé. L'extension de Condor à grande échelle n'a d'ailleurs pas été proposée par ses auteurs sous forme directe, mais hiérarchique : soit comme un réseau P2P de contrôleurs Condor [71], qui s'échangent du travail, soit en s'interfaçant via une grille [179].

3.2.2 Systèmes de localisation et de mise en relation

L'autre grande tendance des systèmes CG-P2P est l'étude de la localisation pour elle- même. L'origine de leur célébrité est bien connue, dans les environnements de partage de fichiers, mais la problématique - localisation d'un objet associé à une clé dans un système à très grande échelle et très volatil - est générale.

L'expérience à grande échelle (Napster) a montré que des solutions très classiques, à base d'architectures hiérarchiques, peuvent être dimensionnées pour satisfaire les demandes de d'utilisateurs. Dans un domaine différent, le système de distribution de contenus Akamai présente également une architecture hiérarchique, quoiqu'à une échelle de distribution supérieure. Dans ces environnements, la localisation est réalisée par un ensemble de brokers,

en petit nombre, et supposés fiables, donc avec une tolérance aux fautes (peu fréquentes) fondée par exemple sur du transactionnel. Seule la demande/fourniture du service (fonctions client et worker) subit les contraintes de passage à l'échelle et de volatilité. Ce type d'architecture est donc proche des systèmes de calcul global, bien que pour un service de nature différente. Une différence importante est cependant que les systèmes de calcul global ont souvent un client unique, ce qui contribue largement à assurer la sécurité des participants, alors que ces environnements symétrisent clients et workers : les ressources sont de la même nature que les demandes (typiquement des fichiers).

L'intérêt s'est ensuite porté vers des systèmes complètement symétriques, où les fonctions de brokers sont aussi prises en charge par des participants très distribués et volatils. Le cœur du problème est alors la définition d'un protocole de placement et de localisation qui assure les trois propriétés suivantes.

- Equilibrage de charge entre les participants.
- Performance de l'algorithme de découverte : en latence et en capacité à découvrir un item s'il est présent.
- Tolérance à la volatilité : il s'agit d'une part de la robustesse fonctionnelle de l'algorithme, d'autre part du coût de gestion de la volatilité pour maintenir cette robustesse.

Les systèmes de seconde génération, typiquement Gnutella [119], ont ciblé très prioritairement le troisième objectif, la tolérance à la volatilité, et réussi à déployer effectivement des systèmes fonctionnels en agrégeant à grande échelle des PC individuels, y compris domestiques (donc avec une connexion très intermittente). A l'inverse, les systèmes de troisième génération (Chord, CAN, Tapestry,...) [80, 158, 202] ciblent le premier et le deuxième objectifs, l'équilibrage de charge et la performance.

3.2.3 Architectures P2P symétriques

L'origine de l'intérêt pour les architectures complètement symétriques peut paraître anecdotique : la distribution à grande échelle des services de mise en relation les protège lorsque l'application est illégale (partage de fichiers musicaux). Elle revisite cependant la problématique classique de l'élimination du point de défaillance unique. De ce point de vue, ce type d'architecture est un candidat pour un système de distribution des calculs, même s'il n'a pas été expérimenté pour l'instant dans ce cadre. Comme on le

verra dans la suite, l'architecture du système XtremWeb est centralisée, avec une vocation à se hiérarchiser. Cette section vise à présenter des éléments de justification - à posteriori - de ce choix.

Extensibilité

Les systèmes de seconde génération peuvent se caractériser comme des réseaux instructurés : la topologie du réseau n'est pas prédéfine, mais émerge dynamiquement comme résultat de l'activité de localisation, avec une structure proche de graphes petit monde (*small world*) [119, 133, 129, 53]. Le protocole de localisation le plus largement expérimenté, l'inondation, est robuste (sans garantir cependant la non-fragmentation du réseau). Ce protocole est en principe peu extensible ; les expériences montrent qu'une fraction non négligeable du débit Internet est consacrée à la gestion de requêtes et de connexion, indépendamment du transfert des données [162]. La question est la possibilité de faire évoluer ce protocole sans renoncer à sa robustesse. Plusieurs travaux ont proposé d'exploiter la structure émergente, en limitant le facteur de branchement de l'inondation, par promenades aléatoires parallèles [4, 99]. Il a cependant été montré que les nœuds à fort degré tendent alors à être surchargés [137]. Des solutions à base d'auto organisation basées sur l'ajustement à la capacité par des algorithmes locaux sont expérimentées [138] .

Les systèmes de troisième génération sont organisés comme des réseaux structurés, avec des topologies classiques (cycle plus hypercubes pour Chord, hypercube pour Tapestry, tore pour CAN), sous-peuplées, d'où un routage incluant des tables fournissant l'identifiant et l'adresse des voisins topologiquement les plus proches. Dans un contexte statique, sans départ ni arrivée de participant, et pour un système parfaitement homogène en capacité réseau et processeurs, les architectures de troisième génération sont optimales en répartition de charge, puisqu'elles hachent les données sur les sommets du réseau, et fournissent des algorithmes de localisation en $O(\log N)$, N étant la taille du réseau.

L'hypothèse d'homogénéité de l'infrastructure matérielle, correcte pour une machine parallèle, n'est pas soutenable pour un système P2P. La symétrie complète conduit à des échanges inutiles à longue distance, qui pénalisent lourdement les performances . L'évolution proposée du système Tapestry vers Brocade [203] propose ainsi de créer des super nœuds, qui agissent comme des gateway entre des réseaux locaux, avec routage local distribué, routage

distribué entre super nœuds, mais un routage direct vers le super nœud gateway du domaine lorsqu'on sort du domaine. Comme les gateway doivent être des machines qui ont un bon accès WAN et une bonne capacité de service du sous-réseau local, on retombe sur une structure hiérarchique bien classique. Et cette hiérarchisation impose deux types d'adressage, autrement dit ne s'inclut pas naturellement dans le système de routage P2P.

Une autre étude [55] démontre que la structure P2P, avec un algorithme de routage en Log, est intrinsèquement moins efficace qu'une structure centralisée (ou modérément distribuée) correctement dimensionnée. Il s'agit de la création d'un système de DNS coopératif basé sur CHORD. Les résultats de simulation pour un système de taille modeste, 1000 nœuds, fournissent une latence 8 fois plus grande que celle du DNS standard. Ces résultats de simulation sont à prendre avec prudence, car la petite taille du réseau a conduit les auteurs à désactiver les mécanismes de cache. Le facteur 8 a deux origines. D'une part, chaque requête demande log(1000) RPC, alors qu'un système de DNS conventionnel en demande typiquement 2, ce qui fournit un facteur 5. Le reste de l'écart est probablement dû au non-respect de l'hétérogénéité de l'infrastructure vue ci-dessus.

Il semble donc que les systèmes P2P complètement symétriques de troisième génération ne soient extensibles qu'à une échelle moyenne, en taille et en homogénéité.

Tolérance aux fautes

L'avantage majeur des systèmes de seconde génération par rapport aux systèmes centralisés est l'élimination du point de défaillance unique, et plus généralement la robustesse. Cette robustesse est atteinte comme sous-produit des actions de localisation, qui construisent le réseau tout en effectuant la recherche des données. Pour les systèmes de troisième génération, un protocole de maintenance séparé doit assurer la réparation permanente de la topologie en présence d'arrivées/départs inopinés. Pour le réseau le plus simple, CHORD, on a montré qu'il existe une topologie dégradée stable, avec un coût de maintenance et par arrivée/départ $\Omega(\log^2 N)$ [134]. Cependant, cet algorithme est synchrone, au sens où il est possible de définir des phases d'exécution d'un algorithme de stabilisation. Dans un contexte complètement asynchrone, il semblerait que le réseau puisse dégénérer vers un état quelconque ; dans ce cas, le protocole synchrone de retour à l'état idéal a un coût prohibitif en $O(N^2)$.

L'étude des algorithmes de stabilisation des systèmes P2P complètement symétriques de troisième génération commence seulement. Il semble cependant que ces systèmes, qui dérivent d'un concept originel statique - sans arrivée ni départ -, soient surtout adaptés à gérer une situation où les modifications du réseau puissent être fréquentes, mais pas inopinées, et où les évènements inopinés restent très exceptionnels.

Chapitre 4

Contributions à XtremWeb

Le projet XtremWeb, qui a commencé à l'automne 99, vise à réaliser une plate-forme d'expérimentation du calcul global et Pair à Pair hautes performances. Sa conception initiale et sa problématique scientifiques ont été présenté très largement dans les articles [p2,p27-p29], dans l'HDR de Franck Cappello [37], et les aspects plus récents dans la thèse de Gilles Fedak [89]. Ce chapitre exposera donc seulement nos contributions.

- Sur les choix d'architecture, dans la première période du projet (99-01)
- Sur un modèle d'analyse de performances.
- Sur la réalisation d'une application d'XtremWeb à une expérience d'astrophysique, dans la période suivante.

4.1 Un système de calcul global

4.1.1 Objectifs

XtremWeb veut être expérimental à deux titres. D'une part, il doit fournir une plate- forme logicielle d'expérimentation des propriétés intrinsèques et de l'architecture des systèmes de Calcul Global, et pouvoir évoluer vers le Pair à Pair. D'autre part, il doit permettre le déploiement d'expériences applicatives réelles.

Le premier objectif pousse à une organisation aussi généraliste et versatile que possible, alors que le second conduit à une approche monolithique, qui exploite complètement les particularités des infrastructures logicielles choisies et des applications visées. Nous en citerons deux exemples.

- Dans le contexte d'une expérience dédiée, où seule l'exécution de codes contrôlés à priori est possible, les problèmes de sécurité des participants sont limités à garantir que la distribution du code et des paramètres ne peut pas être falsifiée, au niveau de l'entité distributrice et au cours du transfert. Au contraire, dans le contexte d'un système où les fournisseurs de ressources et les demandeurs sont interchangeables et anonymes, la sécurité des participants est un problème critique.
- Au niveau de l'architecture interne d'XtremWeb, le choix de la granularité de neutralité par rapport aux outils logiciels, en particulier le langage de développement et les protocoles de communication. Même si le problème paraît technique, voire technologique, il peut conditionner l'intégration des systèmes de Calcul Global dans le contexte général des grilles. Si XtremWeb était à ré-écrire aujourd'hui, serait-il souhaitable de le réaliser à partir de Web Services ?

XtremWeb cible spécifiquement les hautes performances, donc les applications de calcul intensif, qui sont d'ailleurs très souvent aussi data-intensives. Le modèle de calcul visé initialement est celui d'un flot de tâches indépendantes non interactives, donc l'équivalent à grande échelle d'un système de traitement par lots. En revanche, XtremWeb se veut généraliste : l'architecture doit pouvoir servir simultanément et équitablement de multiples applications, même si la création d'un flot de tâches suffisant pour justifier son utilisation provient naturellement d'applications multiparamètres (exécution du même code sur un grand nombre d'entrées). Une conséquence est la nécessité de supporter l'exécution de codes natifs, avec une environnement d'exécution non trivial et contraint (version d'OS, mémoire disponible, accès à une arborescence de fichiers)

4.1.2 Architecture

Deux caractéristiques de l'architecture d'XtremWeb sont l'asymétrie et le modèle pull.

Asymétrie Comme la plupart des systèmes de calcul global, l'architecture d'XtremWeb est inspirée du type classique 3-tier, appliquée au calcul, avec trois types de fonctionnalités : Client, Coordinateur et Worker. L'entité coordinateur assure deux types de services : d'une part la mise en relation des demandes de ressources par les clients et des fournitures de ressource par les workers, donc en particulier le place-

ment/ordonnancement ; d'autre part la supervision de l'exécution après allocation de ressources, en particulier la ré- allocation des travaux perdus sur faute.

L'asymétrie ne réside pas dans ces fonctionnalités, qui sont inévitables, mais dans les hypothèses différentes sur le modèle de faute et l'environnement de ces entités. L'entité worker peut disparaître inopinément à tout instant, et aucune hypothèse ne peut être faite sur la qualité (en particulier le délai) de sa connexion avec l'entité coordinateur. L'entité coordinateur doit être exempte de fautes, au sens où sa défaillance entraîne la défaillance de l'ensemble du système.

On retrouve la même asymétrie du point de vue de l'environnement. Les contraintes sur celui des workers doivent être minimales, dans le cadre d'un partage où l'utilisateur légitime ne doit pas être perturbé. En particulier, le worker peut être derrière un firewall, ne pas posséder d'adresse IP fixe etc.

Modèle pull La conception du protocole de communication entre les entités workers et l'entité coordinateur tire la conséquence de cette asymétrie en imposant que toute communication soit à l'initiative du worker : le coordinateur ne peut pas obtenir d'information sur le worker à sa convenance, mais seulement lorsque le worker entre en contact avec lui. Il s'agit donc d'un protocole de type *soft-state* [157]. Ce point est très fortement lié avec le mécanisme de tolérance aux fautes. Il s'agit d'un choix architectural. Du point de vue technique, le problème des firewall peut être résolu en encapsulant les communications qui iraient d'un coordinateur vers un worker dans des protocoles adaptés [84]. La solution pull est cependant largement préférable : l'encapsulation d'une part pose des exigences sur l'environnement des workers, d'autre part est orthogonale à la gestion des fautes. L'implémentation actuelle utilise, soit des RPC synchrones (XML-RPC, Java RMI) découplés, c'est à dire pris en charge par un flot d'exécution qui gère les communications sur le worker, soit un flot TCP [89].

On notera qu'on a parlé d'entités : c'est l'occasion de définir le degré de généralité de la première version d'XtremWeb. L'architecture est clairement centralisée. En revanche, la nature des entités worker et coordinateur est relativement peu contrainte. Du point de vue Worker, le système est complètement indifférent à la nature de la tâche de calcul, qui peut être parallèle, en modèle passage de messages comme en multithread : une ma-

chine multiprocesseur ou une grappe peut être le support matériel de l'entité worker. Il faut noter qu'à cette étape, le système lui-même ne supporte que le parallélisme trivial (exécutions indépendantes) : chaque worker peut être parallèle, mais leurs exécutions ne sont pas couplables. L'exécution parallèle sur un système volatil est traitée dans le chapitre 6.

4.2 Tolérance aux fautes

4.2.1 Un système Crash-Only

Le modèle de faute considéré est celui de l'arrêt sans préavis, à partir duquel il faut récupérer un comportement cohérent de l'ensemble du système, c'est à dire :

- au moins une exécution réussie de la tâche de calcul ;
- donner à voir au monde extérieur une seule exécution : le calcul peut n'être pas idempotent dans le contexte du client d'XtremWeb, donc celui-ci doit recevoir un et un seul résultat.

Par ce choix, XtremWeb diffère des systèmes de gestion de lots avec migration sur grappes (Condor), pour lesquels le départ sans préavis est un pire cas supposé relativement peu fréquent. Le préavis de départ permet alors d'organiser la récupération de l'exécution du processus par sauvegarde de son image (checkpoint) et migration.

Plus précisément, XtremWeb met en œuvre une stratégie crash-only [36]. Ce type de stratégie a été défini, indépendamment des travaux sur les grilles, dans le contexte de systèmes distribués basés sur des logiciels commerciaux (serveurs Web distribués [9, 40], acquisition et traitement de données satellite [57]), par opposition avec les stratégies de tolérance aux fautes qui visent, soit le retour à un fonctionnement normal, soit le retrait en ordre (clean shutdown) qui préserve une partie significative de l'état du système.

Dans une architecture crash-only, lorsqu'une situation anormale est détectée, le composant logiciel mis en cause est contraint à l'arrêt complet suivi d'un redémarrage. Les fautes du système, éventuellement transitoires, sont donc forcées à un comportement de fautes fatales (composants fail-stop). Le schéma de fautes du système initial, basé sur des composants logiciels hétérogènes et opaques, serait extrêmement complexe à modéliser. Le système transformé devient relativement plus simple, la modélisation ne devant plus porter que sur les dépendances de redémarrage. La reconfiguration vers un système fonc-

tionnel s'effectue par redémarrage hiérarchique des composants.

Les expériences rapportées [35, 143] ont monté une amélioration remarquable de la disponibilité par la stratégie crash-only, d'un : facteur 4 pour l'expérience Mercury et de 50% pour le serveur distribué PRESS. L'intuition suivant laquelle la sauvegarde d'une partie de l'état du système devrait permettre une récupération plus rapide ne s'applique donc pas à la classe de systèmes

> distinguished by large scale, stringent high-availability requirements, built from many heterogeneous components, accessed over standard request-reply protocols, serving workloads that consist of large numbers of relatively short tasks that frame state updates and subjected to rapid and perpetual evolution [35].

La partie fonctionnelle de cette description s'applique clairement aux systèmes de calcul global. Dans le cas d'XtremWeb, dans sa version initiale, l'entité Coordinateur étant supposée exempte de faute, le système dont la disponibilité doit être maximisée est l'ensemble d'une part des composants logiciels qui s'exécutent sur workers, d'autre part des communications entre workers et Coordinateur. Les éléments de logiciel qui s'exécutent sur les workers constituent typiquement de composants hétérogènes, par leur support matériel et logiciel de base, à comportement très dynamique et également potentiellement très hétérogène.

Pour forcer les fautes vers des fautes fatales, deux principes complémentaires ont été utilisés :

- les états potentiellement transitoires du système sont de type soft-state
- les états persistants sont gérés par un environnement spécifique implémentant une sémantique ACID .

4.2.2 Soft-State

Le concept de soft-state a été initialement défini dans le contexte des réseaux [50]. L'état d'un composant du système est conservé par un agent, agissant comme un cache, qui annule l'état à l'expiration d'un délai prédéfini, sauf si cet état à été rafraîchi par un message de mise à jour provenant d'un autre agent du système. La source de l'état transmet donc des messages, qui sont susceptibles d'être perdus, mais pas altérés, vers un ou plusieurs récepteurs qui possèdent ainsi une copie, éventuellement périmée, de cet état. Un timer associé à l'état, est redémarré à la réception d'un message, qui

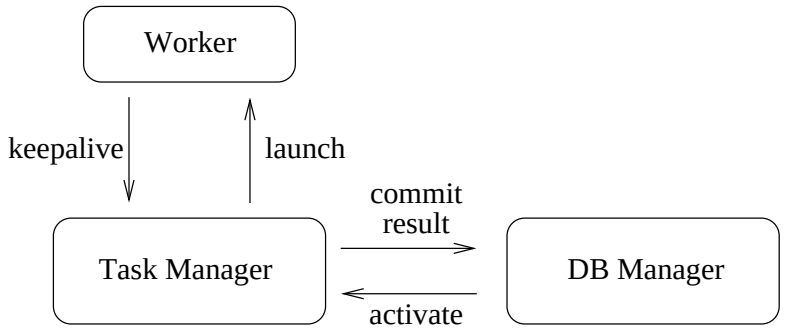


FIG. 4.1 – Architecture Soft-State d'XtremWeb

convoie éventuellement une valeur de l'état. L'expiration du délai entraîne la destruction de l'information sur l'état, et éventuellement le démarrage d'une action permettant d'obtenir une copie nouvelle.

Cette conception, qui a été utilisée massivement dans des protocoles de communication basés sur IP, est considérée comme centrale dans les architectures de grilles. L'initiative Open Grid Service Infrastructure (OGSI) [86], qui a récemment évolué vers la spécification Web Service Resource Framework (WS-RF) [75] vise la spécification d'un environnement distribué général permettant des services stateful au dessus des Services Web, qui sont orientés programmation stateless. OGSI introduit plusieurs éléments reliés à ce concept. Le plus important est la notion de service transitoire (transient service). L'entité qui demande a création d'un service (le client en terminologie OGSI) spécifie une durée de vie minimale du service (via une fenêtre de dates absolues, à l'intérieur de laquelle le service de création choisit une date de terminaison). Le client peut ensuite entretenir la vie du service par des messages de keepalive (*GridService : :RequestTerminationAfter*). Les éléments de son état (*serviceData*) qu'un service souhaite rendre visibles à l'extérieur possèdent également des attributs de validité temporelle (*goodFrom*, *goodUntil*, *availableUntil*) ; enfin, une interface pour la notification d'évènements reliés au cycle de vie des instances de services ou des serviceData est définie.

Ce concept a été implémenté dans l'architecture d'XtremWeb 1.0 (fig. 4.1). L'entité Coordinateur a deux composants principaux, le *TaskManager* et le *DBManager*. Le TaskManager gère les instances dynamiques, et donc éventuellement volatiles, des travaux à exécuter, sur un modèle soft-state : les workers envoient périodiquement des messages de keepalive, dont l'absence,

à l'expiration du délai de timeout, définit le travail supporté par ce worker comme perdu ; la tâche correspondante doit être relancée sur un worker disponible. La gestion du placement/ordonnancement des tâches sur les workers relève également du TaskManager.

L'équivalent de ce dispositif en architecture OGSI pourrait être un service Scheduler qui reçoit les requêtes de travail des workers et crée (ou recycle) sur une telle requête un service de surveillance de la tâche dévolue à ce worker. Le scheduler lui-même peut souscrire à la notification des modifications de validité temporelle du service de surveillance, qui s'autodétruit (ou se réinitialise) lorsque sa durée de vie expire.

L'architecture logicielle actuelle d'XtremWeb ne suit pas exactement ce modèle : le service de gestion d'une tâche est partagé entre plusieurs composants logiciels. Il ne serait cependant pas très difficile de le faire évoluer dans cette direction. Le principal intérêt est de permettre la distribution des ces services. L'architecture actuelle présente en effet un goulet d'étranglement, le débit réseau du coordinateur. Le coordinateur doit supporter la charge liée au lancement des tâches (données d'entrée et résultats), et les keepalive. La distribution des services de gestion des tâches lève en partie cet obstacle à l'extensibilité, en distribuant la gestion des keepalive. Si cette distribution s'effectue sur des machines fiables, le mécanisme de notification peut être léger, typiquement à l'initiative des services de gestion ; c'est d'ailleurs explicitement l'intention de la spécification OGSI :

> The treatement of subscriptions as Grid services instances allows them to be managed as other Grid services. An OGSI implementation might well be expected to use specialized, more lightweight implementation techniques for subscription than for other Grid service instances [86].

Dans une perspective d'évolution d'XtremWeb vers des systèmes P2P, la seule distribution des services de gestion de tâches sur des machines elles-mêmes volatiles ne résout pas le problème d'extensibilité : la notification ne peut alors reposer que sur un nouveau mécanisme de keepalive, qui soit sera peu efficace s'il examine trop peu souvent l'état des services, soit également consommateur de ressources réseau.

Le DBManager correspond au stockage permanent spécialisé des systèmes crash-only. Il permet la récupération après une défaillance matérielle du Coordinateur, ou la détection d'une incohérence grave dans le fonctionnement du TaskManager suite à un bug logiciel.

Le seul modèle de faute des workers considéré initialement est celui des fautes fatales des workers : lorsque le processus en charge de l'exécution d'une tâche disparaît, le travail effectué est perdu. L'arrivée d'un keepalive après le timeout est ainsi considéré comme une re-connexion du worker.

Il faut souligner que ce mécanisme, par définition, ne fait pas d'hypothèse sur la permanence de la connexion des workers. Les mesures sur des systèmes de partage de fichiers [198], ainsi que des mesures générales de durée de connexion [141], on montré que la fréquence des connexionsdéconnexions est élevée chez les utilisateurs individuels (même si cette situation est probablement amenée à évoluer rapidement). Un calibrage adéquat du timeout permet de cibler également cette ressource de calcul sans modifier le modèle précédent. Un tel calibrage repose sur la possibilité de prédire individuellement, au moins grossièrement, la fréquence des déconnexions des participants [141, 121, 120, 122].

4.3 Un modèle de performances

4.3.1 Motivation

La stratégie crash-only fournit un modèle d'exploitation léger et cohérent, mais qui n'est intuitivement pas approprié au traitement des calculs "longs". Il existe un choix alternatif à la stratégie crash-only, qui est la sauvegarde (checkpoint) ; dans le cadre de calculs multiséquentiels, la sauvegarde se limite à l'enregistrement fiable d'une image mémoire, par exemple périodique, ce qui permet de redémarrer l'exécution à partir de la dernière sauvegarde. Dans les systèmes locaux, à la Condor, la fiabilité de la sauvegarde repose sur celle du système de fichiers local. Cette solution n'est pas extensible à l'échelle d'un système de calcul global ; la sauvegarde devrait reposer sur un système de fichiers distribués [81] ou de stockage réparti [56]. Dans le premier cas, le système léger et peu intrusif de Calcul Global se trouve partager des caractéristiques d'un système de grille classique , avec des problèmes de localisation des données, de localité des calculs par rapport aux données etc., qui requièrent un système d'information lourd. La seconde solution est encore très conjecturale. Quoiqu'il en soit, la sauvegarde a un coût important, en termes d'espace de stockage et/ou de débit réseau.

Une version légère correspond à une sauvegarde locale sur le worker ; le worker peut alors fractionner une exécution longue, à condition bien sûr que

sa fréquence de déconnexion ne soit pas trop élevée pour permettre au calcul d'avancer. Une version "super-légère" correspond à une absence de sauvegarde, le seul acquis par rapport à un démarrage à froid étant la présence du fichier d'entrée [5]. La complexité logicielle de ces systèmes est cependant supérieure à celle du système crash-only : essentiellement, le système va devoir gérer des tâches clonées, et dont les dates de démarrage ne sont pas nécessairement proches. Bien entendu, dans le système crash-only, on a également la possibilité d'exécutions simultanées correspondant à une même tâche, mais une seule est répertoriée comme légitime : les autres pourraient correspondre par exemple à un signal keepalive très retardé, et ont été supprimées du système d'information du coordinateur par timeout, donc peuvent être éliminées sans ambiguïté si elles retournent.

Au moins dans un modèle relativement simplifié du comportement des tâches, on peut donner un modèle quantitatif d'un système de calcul global purement crash-only. On peut également modéliser l'influence d'une stratégie complémentaire, la réplication des tâches. Cette modélisation, décrite dans les sections qui suivent, montre que la stratégie crash-only a un comportement dans le détail complexe, mais qui aboutit à la possibilité d'un dimensionnement assez précis du système.

4.3.2 Modélisation

On se limite ici au cas de tâches séquentielles. La première question est le type de modèle significatif pour le Calcul Global. Très schématiquement, on peut considérer, soit un contexte d'ordonnancement, soit un contexte de files d'attente. Dans le premier cas, on cherche à optimiser le temps de complétion (makespan) d'un lot de tâches considéré isolément ; dans le second, le système est soumis à un flot d'entrée permanent ; on peut alors chercher à optimiser et le temps de complétion de chaque tâche et le débit du système. L'utilisation typique d'un système de calcul global a souvent été le multiparamètres, portant sur une durée très longue par rapport au temps d'exécution des tâches individuelles. Le critère de performance est alors le débit du système. Dans une perspective plus générale, le système de Calcul Global est utilisé comme service banalisé ; un critère important est alors le temps de complétion d'une tâche. Dans les deux cas cependant, il semble que le système doive être plutôt considéré comme en fonctionnement permanent.

Les paramètres de la performance sont essentiellement :

– le nombre de machines disponibles (en les supposant homogènes) ;

- le processus temporel des évènements fautes, par rapport à chaque tâche ;
- la durée des tâches (hors faute) ;
- la période des keepalive.

A notre connaissances, les études analytiques sur les systèmes à défaillances inopinées ont porté sur l'algorithmique de la sauvegarde : étant donnés une loi statistique d'apparition des fautes sur une machine, et un coût (constant) pour chaque sauvegarde, définir un échéancier (schedule) de sauvegardes. Il s'agit d'études déjà anciennes [51], qui ont été reprises et étendues dans [22, 165, 163, 164]. Ces derniers travaux visent plutôt le contexte des *tâches malléables*, où un surcoût fixe est associé à chaque fragment de tâches envoyé à l'exécution. Dans les deux cas, l'échéancier doit maximiser la quantité de travail effectué *avant la première faute*.

Le point de vue ici est très différent : on ne considère pas une machine isolément, dont le processus de fautes serait statistiquement prévisible, en particulier à partir de son comportement antérieur. On considère une tâche isolée (en 4.3.3), ou un ensemble de tâches (en 4.3.4), dont les supports d'exécution successifs sont des machines choisies aléatoirement. Le premier problème est alors de formuler une hypothèse raisonnable sur le processus temporel (t_p) défini par la suite des instants des fautes *sur les machines successives*. Si on suppose que les comportements des machines ne sont pas corrélés, sont invariants dans le temps, et que parmi celles-ci on effectue un choix aléatoire, une modélisation naturelle est un processus de Poisson.

La validité de cette hypothèse est certainement liée à la taille et surtout à l'extension géographique et organisationnelle du pool de machines gérées par un coordinateur : à l'extrême, des machines très fortement corrélées (machines d'une salle d'enseignement, ou d'une entreprise) auront un comportement quasi-identique, au moins à une échelle grossière. En revanche, si les machines sont indépendantes et choisies aléatoirement, il est raisonnable de supposer que le processus des fautes successives est sans mémoire, ce qui conduit à un modèle Poissonnien. C'est d'ailleurs par exemple l'hypothèse adoptée dans l'analyse de performances d'un des systèmes classiques de DHT [134]. Il faut souligner que cette hypothèse n'implique en aucune façon que le processus des fautes de chacune des machines soit Poissonnien, ce qui est très probablement faux. La caractérisation statistique du comportement d'une machine, en particulier du point de vue de la charge, a été largement étudiée [61, 6, 30, 62, 161]. Ces études convergent vers un comportement auto-similaire, probablement instable dans le temps.

Le processus temporel qui décrit les fautes subies par une tâche particulière est donc supposé Poissonien, i.e. délais inter-arrivée indépendants et de densité commune $\lambda e^{-\lambda t}$, où λ est le taux de fautes. On suppose également les tâches de même durée (hors faute), qui définit l'unité de temps (donc les tâches sont de durée 1).

4.3.3 Latence

La latence est le temps de complétion d'une tâche isolée. Autrement dit, on considère un système avec des ressources (processeurs) suffisantes pour servir immédiatement toute tâche qui se présente ; cette hypothèse sera reconsidérée dans par la suite, mais permet de borner supérieurement les performances.

Sans réplication

Soit α la période des keepalive ; l'unité de temps étant le temps d'exécution (sans faute) d'une tâche, en supposant pour simplifier $1/\alpha$ entier, la fréquence définit le nombre de keepalive pendant une exécution sans faute.

Une exécution peut alors être modélisée par des tirages successifs indépendants, chacun correspondant à une période, avec un résultat 0 si une faute intervient pendant la période et un résultat 1 sinon. La probabilité p de tirer un 1 est $e^{-\lambda\alpha}$. L'exécution se termine à la fin de la première séquence de $N = 1/\alpha$ 1 consécutifs. Soit τ_N la longueur de la séquence complète (par exemple, pour la séquence 100111, $\tau_2 = 5$ et $\tau_3 = 6$). La latence moyenne $\overline{T_1}$ est :

$$\overline{T_1} = \alpha E(\tau_{\alpha^{-1}})$$

On a :

$$\tau_N = 1 + \tau_{N-1} + (1-p)\phi_N,$$

où ϕ_N est une variable aléatoire de même loi que τ_N. En effet, après $N-1$ succès :

– si succès, la séquence est complète, donc $\tau_N = 1 + \tau_{N-1}$;

– si échec, le calcul de la séquence redémarre, donc $\tau_N = 1 + \tau_{N-1} + \phi_N$

D'où $E(\tau_N) = \frac{1+E(\tau_{N-1})}{p}$; en résolvant la récurrence, et en utilisant la définition de p, on obtient

$$\overline{T_1} = \alpha \frac{e^{\lambda} - 1}{1 - e^{-\lambda\alpha}}.$$

Le temps d'exécution moyen est donc une fonction

- Exponentiellement croissante du taux de fautes λ (fig. 4.2 a) ;
- Décroissante de la fréquence des timeouts (fig. 4.2 b) ; mais cette décroissance est seulement quasi linéaire ; en effet pour α petit, on a $(1 - e^{\lambda\alpha})^{-1} \approx (\lambda\alpha)^{-1}(1 + \lambda\alpha/2)$.

Le plus important est qu'il n'est pas possible d'obtenir une qualité de service définie à priori par la seule adaptation de la fréquence des keepalive. La borne inférieure du temps d'exécution correspond à $\alpha = 0$, c'est à dire la notification immédiate de la faute ; la borne supérieure correspond à $\alpha = 1$, c'est à dire pas de keepalive. On a

$$\overline{T}_1^{\min} = \lambda^{-1}(e^{\lambda} - 1); \qquad \overline{T}_1^{\max} = e^{\lambda}$$

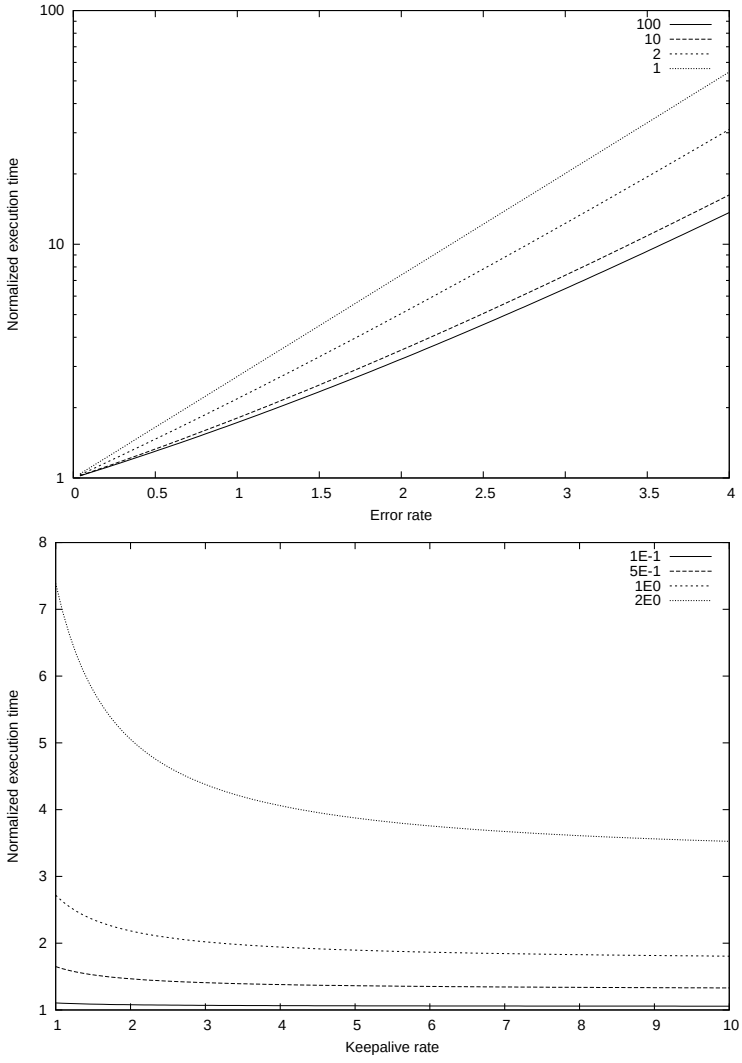
La qualité de service ne peut donc provenir que d'une prédiction précise de la fréquence attendue des fautes. Un placement adaptatif devrait allouer les tâches en fonction du produit du temps d'exécution estimé de la tâche et de la fréquence prédite des fautes ; ce produit correspond, dans l'analyse précédente, au seul paramètre λ, le temps d'exécution étant normalisé.

Cette approche n'est pas totalement crédible : la dépendance exponentielle en λ implique une sensibilité excessive à la prédiction. L'amélioration de la qualité de service peut alors être recherchée par *réplication des tâches*. Une étude expérimentale (par simulation) sur ce sujet a exhibé un comportement complexe [122]. Ce qui suit présente un modèle analytique.

Latence avec réplication

Chaque tâche est répliquée n fois, et on suppose une détection immédiate des fautes. Plus précisément, lorsqu'une tâche demande l'exécution, n copies de cette tâche sont lancées sur les processeurs $1, \ldots, n$; l'examen du statut de la tâche (succès ou échec) n'intervient que lors de la complétion d'une des copies, ou de la détection d'une faute sur le dernier processeur actif qui exécutait une des copies ; lorsque cette faute est détectée, impliquant qu'aucune des copies n'a réussi à s'exécuter, un nouveau lot de n copies est relancé. Soit t_k^i l'instant de la première faute sur le processeur k ($1 \leq k \leq n$) à la $i$ème tentative. On fait trois hypothèses supplémentaires :

- les processus de faute sont sans mémoire entre les tentatives, i.e. pour tout i, les t_k^i suivent la même loi t_k ;
- les t_k sont des variables aléatoire indépendantes, *i.e.* les fautes sont indépendantes entre processeurs ;
- les t_k suivent une loi exponentielle de paramètre λ.



(a) (b)

FIG. 4.2 – Temps moyen d'exécution en fonction (a) du taux de fautes pour diverses valeurs de la fréquence des keepalive (b) de la fréquence des keepalive pour diverses valeurs de la fréquence du taux de fautes

Soit $Y = \max\{t_k; k = 1 \dots n\}$. La distribution de Y est définie par :

$$P(Y \leq t) = \prod_{k=1}^{n} P(t_k \leq t) = (1 - e^{-\lambda t})^n$$

Sous les deux premières hypothèses, on peut modéliser le temps d'exécution par une v.a. T_n qui satifait la relation :

$$T_n = 1_{\{Y>1\}} + 1_{\{Y\leq 1\}}(Y + T'_n)$$

où T'_n est indépendante de T_n et a la même loi : si sur au moins un processeur, la première faute n'intervient qu'après le temps 1, la durée d'exécution est 1 ; sinon, l'exécution est relancée à partir de l'instant de la dernière faute. En prenant les espérances des deux membres, on obtient :

$$\overline{T_n} = 1 + \frac{E(Y 1_{\{Y\leq 1\}})}{P(Y > 1)};$$

puis, après quelques calculs,

$$\overline{T_n} = \frac{1}{\lambda(1 - (1 - e^{-\lambda})^n)} \sum_{k=1}^{n} \frac{(1 - e^{-\lambda})^k}{k}.$$

L'intérêt de ce résultat est que la réplication permet d'assurer en principe une qualité de service arbitrairement bonne : en utilisant

$$\sum_{k=1}^{\infty} x^k/k = -\ln(1 - x),$$

on montre facilement que T_n tend vers 1 quand n tend vers l'infini.

La fig. 4.3 donne l'allure du comportement de T_n. On voit qu'il y a deux régimes : au début, l'augmentation du taux de réplication fait diminuer dramatiquement la latence ; ensuite, on a un phénomène de rendements décroissants.

L'hypothèse de ressources non bornées, correspond à une situation de vraiment très grande échelle et d'exécution d'un lot de traitements prédéfini. En ajustant dynamiquement le taux de réplication au taux de fautes pour une qualité de service donnée, des configurations avec un taux de fautes très élevé (plus grand que 1) peuvent être exploitées utilement.

Cependant, l'usage réel d'un système de calcul global ou P2P est plus probablement celui d'un flot continu de tâches. Cette situation est étudiée dans la section suivante.

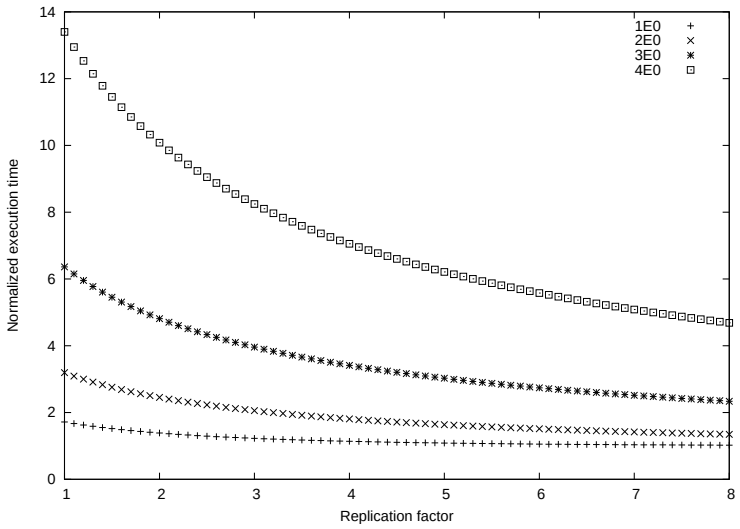


FIG. 4.3 – Temps moyen d'exécution en fonction du taux de réplication pour diverses valeurs du taux de fautes

4.3.4 Dimensionnement

La situation la plus courante est celle de ressources bornées, typiquement pour l'exploitation des cycles inutilisés d'une institution ou d'une entreprise. Le flot d'arrivée des tâches doit être régulé. Chaque tâche relancée, ou dupliquée, alourdit la charge du système, qui peut alors se modéliser comme une file d'attente. La question est alors le dimensionnement du système, en termes de relation entre le nombre de processeurs et débit d'entrée.

- L'arrivée des tâche est supposée Poissonienne de paramètre θ ;
- Le temps de service est donné par la latence isolée ; on a vu que ce temps n'est pas exponentiel.

Le modèle le plus simple est celui d'une file de type M/G/1. Dans le cas général d'une politique de réplication avec un facteur de réplication n, le taux d'arrivée Θ dans cette file doit être le taux d'arrivée des tâches θ multiplié par un facteur qui dépend de n. On donne la valeur n à ce facteur, i.e. $\Theta = n\theta$; ceci correspond à deux approximations. La première revient à considérer que les processeurs alloués pour les répliques d'une même tâche restent alloués à cette tâche, alors qu'un processeur qui vient de faire une faute peut se reconnecter et récupérer du travail nouveau avant que le dernier de ses co-allocataires ait réussi ou échoué. La seconde provient du fait qu'on ne tient

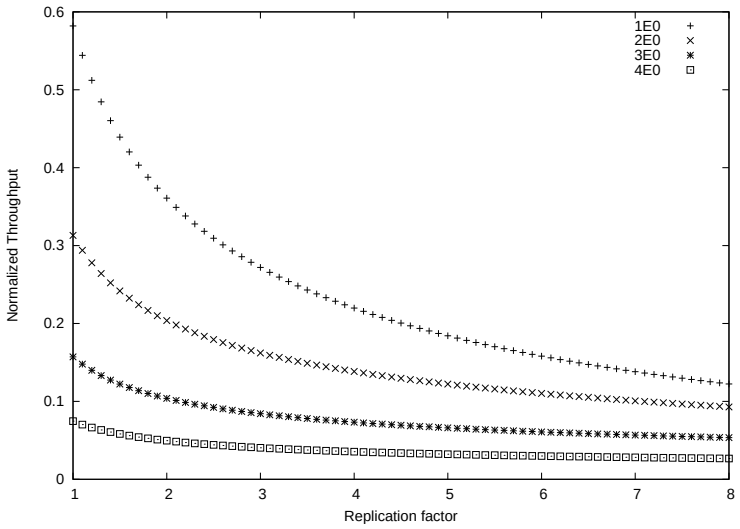


FIG. 4.4 – Débit pour diverses valeurs du taux de fautes

pas compte de l'arrivée simultanée des tâches répliquées (arrivées *burst*).

Les quantités significatives pour la performance sont donc, d'une part le débit d'entrée acceptable par le système, d'autre part le temps de complétion d'une tâche.

Débit

Pour une file d'attente à l'équilibre, le débit de sortie est égal au débit d'entrée et le produit du débit d'entrée par le temps moyen de service (facteur d'occupation) est inférieur à 1. On a donc la contrainte pour chaque file d'attente du système :

$$n\theta\overline{T_n} < 1.$$

L'augmentation du facteur de réplication est toujours défavorable au débit (fig. 4.4) : on vérifie facilement que $n\overline{T_n}$ est une fonction croissante de n.

Soit Φ le débit d'entrée d'un système à P processeurs. Les hypothèses simplificatrices faites ci-dessus permettent de considérer que le système est composé de P/n files d'attentes de type M/G/1. Si la charge est également répartie, on a $\theta = \frac{\Phi}{P/n}$, soit

$$\frac{\Phi}{P} < \frac{1}{n^2\overline{T_n}}$$

Le paramètre $1/n^2\overline{T_n}$ peut s'interpréter de deux façons. Dans un système mono-application, le débit du système est proportionnel au nombre de processeurs que l'application est parvenue à rassembler, et $1/n^2\overline{T_n}$ représente le coefficient de proportionnalité. Dans un système pair à pair idéal, où chacun contribue *en moyenne* autant qu'il consomme, Φ/p est la consommation de chacun, qui est bornée par $1/n^2\overline{T_n}$.

Temps de complétion

On cherche à évaluer $\overline{T}$, le temps moyen passé dans le système par une tâche, incluant le temps de service $\overline{T_n}$ et le temps d'attente avant d'accéder aux ressources de calcul. $\overline{T}$ peut s'évaluer à partir du théorème de Little $\overline{N} = \Theta\overline{T}$, où $\overline{N}$ est le nombre moyen de tâches présentes dans le système. La formule de la valeur moyenne de Pollaczek-Kinchin donne une estimation robuste de $\overline{N}$ pour la queue $M/G/1$:

$$\overline{N} = \rho + \frac{\Theta^2\overline{T_n^2}}{2(1-\rho)},$$

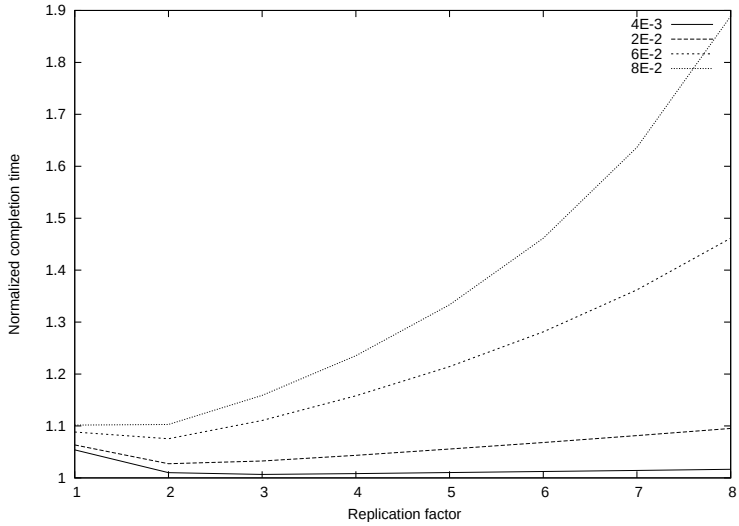
où ρ est le facteur d'utilisation, avec $\rho = n\theta\overline{T_n}$.

Sans réplication, on obtient une formule explicite pour le temps moyen passé dans le système :

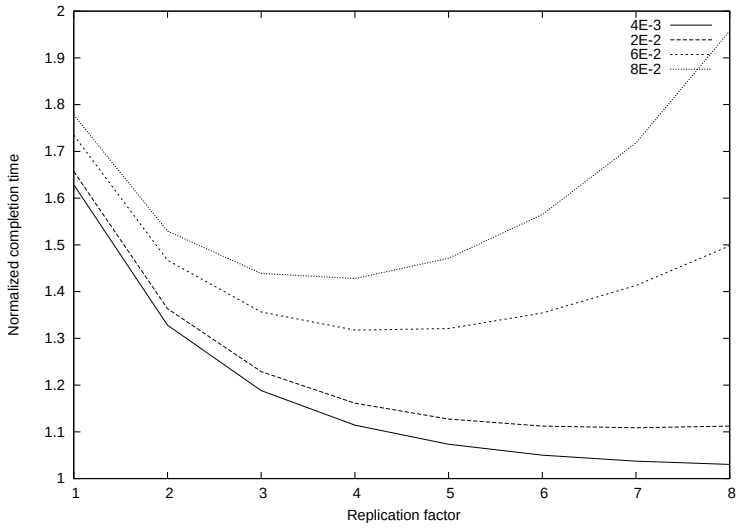
$$\overline{T} = \frac{e^\lambda - 1 - \frac{\lambda\theta}{\lambda+\theta}e^\lambda}{\lambda - \frac{\lambda\theta}{\lambda+\theta}e^\lambda}.$$

Avec réplication, on ne peut obtenir que des expressions contenant des intégrales, qu'il faut évaluer numériquement.

La figure 4.5 donne l'allure du temps de complétion. On observe un régime complexe : pour un taux de fautes faible, la réplication n'est pas productive et peut être très défavorable ; pour un taux de fautes élevé, la réplication peut être productive. Dans les deux cas, le taux d'arrivée des tâches conduit à des comportements différents. Sans surprise, cette évaluation montre que la réplication améliore le débit lorsque le taux de fautes est élevé et que le système est peu chargé. La complexité du comportement indique cependant que l'exploitation de la réplication demanderait un mécanisme d'adaptation dynamique.



(a)



(b)

FIG. 4.5 – Temps de complétion pour diverses valeurs du taux d'arrivée θ : (a) taux de faute faible (0.1) (b) Taux de fautes élevé (0.9)

Chapitre 5

Certification de résultats

Cette section étend [p34] et [p37].

5.1 Introduction

Les systèmes de Calcul Global sont fortement exposés à la corruption des calculs effectués par les collaborateurs. Un très petit nombre de systèmes de Calcul Global ont été effectivement déployés dans un environnement ouvert. Les trois plus connus (distributed.net, SETI@home, decrypthon) ont été victimes d'attaques de type falsification, où des résultats inexacts sont retournés par les collaborateurs. L'inexactitude ne portant que sur la valeur, et non le format, des résultats, la falsification n'est pas détectable immédiatement par le système de calcul global.

Les motifs de falsifications sont multiples. Les simples fausses manœuvres sont évidemment possibles. Dans le cas de SETI@home, qui est bien documenté, certains collaborateurs ont substitué à la fonction de FFT originale une fonction plus rapide, mais qui ne répondait pas aux spécifications de l'application en termes de précision numérique. L'objectif apparent était de traiter un plus grand nombre de jobs, pour obtenir un meilleur classement dans le "hall of fame" de SETI. SETI@home était un système de Calcul Global pur, avec une seule application, délivrée en outre sous forme de binaire et non de code source. Une autre possibilité, dans le cadre du Calcul Global pur, est la volonté de biaiser les résultats. Pour un système de calcul P2P de mutualisation de ressources de calcul, il existe aussi un fort risque de l'analogue du "free riding" des systèmes de partages de fichiers : les collaborateurs

utilisent les ressources, mais n'en fournissent pas.

Enfin, le risque d'*attaque en déni de service* est probablement la plus sérieuse à long terme. L'objectif n'est pas de falsifier les résultats pour eux-même, mais de rendre impossible le fonctionnement du système de calcul global, en effondrant son rendement, ou en détruisant sa crédibilité.

Deux classes de stratégies peuvent être utilisées pour interdire ou limiter la falsification : la prévention a priori ou la vérification a posteriori. La prévention tente de contraindre les collaborateurs à utiliser les logiciels et les fichiers voulus par le client. Au niveau utilisateur, il s'agit des méthodes de cryptage de code, qui tracent l'utilisation du code, par exemple en calculant des signatures d'exécution à divers points du code. Au niveau système, la fidélité du code et des données peut être réalisée par une chaîne de certification dont le garant ultime est matériel (initiative TCPA-Palladium [103], modification du microcode [107]). Outre les problèmes sociologiques majeurs soulevés par la prévention système, aucune des deux méthodes n'est applicable au contexte de logiciel à code source ouvert.

La seconde stratégie, la vérification à posteriori, est celle qui est considérée ici.

La problématique générale de cette section est

- Verification. Il s'agit de définir des méthodes d'évaluation quantitatives de l'exactitude des résultats obtenus lors d'exécutions dans un environnement de type calcul global, où la falsification est possible.
- Environnements d'exécution en présence de falsifications, l'objectif étant un impact aussi faible que possible sur les performances du système de calcul global, et d'évaluer quantitativement cet impact afin de le rendre paramétrable. La vérification constitue donc une infrastructure, que les applications directement, ou l'intergiciel, peuvent exploiter pour construire de la tolérance aux fautes.

Nos premiers travaux ont porté sur la définition d'algorithmes basés sur une méthode de test d'hypothèse en relation avec la programmation dynamique. L'originalité de notre approche, par rapport à des travaux portant sur la certification en calcul global [166, 116], est : d'une part, de tenter une démarche globale d'estimation statistique du taux de falsification, incluant les risques de première espèce (faux positif) aussi bien que ceux de seconde espèce (faux négatif), sans requérir d'information a priori sur la distribution des falsifications ; d'autre part, d'exploiter les caractéristiques vraisemblables du comportement des adversaires dans le cadre du calcul Global pour adapter le coût des algorithmes de vérification à leur comportement.

Le modèle de calcul global ciblé ainsi très général ; en particulier, nous montrerons qu'il est possible de définir des mécanismes efficaces de détection sans faire d'hypothèse sur le schéma de coopération des adversaires.

Ces algorithmes ont été définis et évalués dans la perspective du problème général suivant : définir un modèle d'exploitation et les mécanismes qui l'implémentent qui prennent en entrée les contraintes, d'une part d'exploitation (surcoût de certification), d'autre part de validité (exigence de qualité utilisateur), pour paramétrer l'allocation de ressources d'exécution.

Outre leur application aux systèmes de calcul global stricto-sensu, ces travaux pourraient avoir des applications dans des contextes de collaborations larges, où des simulations numériques très nombreuses sont réalisées par des individus, institutions, etc. divers, et où la chaîne des traitements informatiques n'est pas complètement automatisée. La possibilité d'erreurs de manipulation est alors bien réelle. A ce titre, et indépendamment d'XtremWeb, ce travail est l'objet d'une collaboration avec l'équipe Auger.

5.2 Calcul Global et vérification

5.2.1 Contextes

En supposant le coordinateur exempt de fautes, la corruption peut avoir deux sources : lors de la transmission, ou lors du calcul. Si les collaborateurs sont parfaitement fiables, les corruptions en transmission peuvent être évitées par les techniques classiques de cryptographie, indépendamment de l'application cible. Les Grids classiques fonctionnent sur ce modèle : les entités collaboratrices sont supposées fiables et bien gérées, donc adhèrent à des protocoles de protection de leurs communications.

Au contraire, dans le cas du Calcul Global, nous ferons l'hypothèse qu'aucune garantie n'est fournie à priori sur la fiabilité des collaborateurs. Le système de calcul global est vu comme un service, qui délivre des unités de calcul, les *jobs*, aux collaborateurs, et en récupère les résultats. Au niveau de la vérification, les éventuelles dépendances entre les jobs ne sont pas prises en compte.

L'objectif est de définir une *procédure de test*. Soit $\mathcal{E}$ un ensemble d'objets, muni d'une distance Δ, et $\mathcal{P}$ le sous-ensemble des objets de $\mathcal{E}$ qui vérifient une propriété donnée. Un *test* est un algorithme $\mathcal{A}$ probabiliste qui prend en entrée trois réels positifs (α, β, k), et un objet o de $\mathcal{E}$, et qui vérifie les

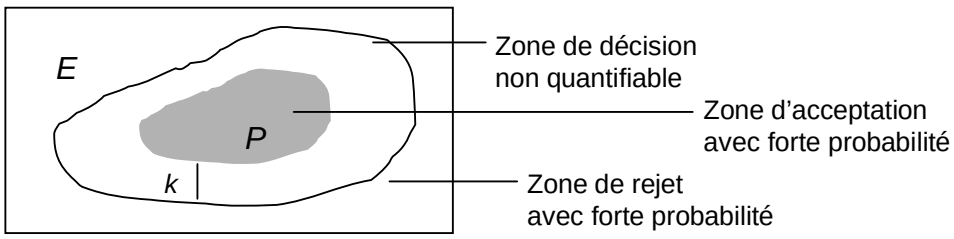


FIG. 5.1 – Procédure de décision

propriétés suivantes :

- si $o \in \mathcal{P}$, alors $\mathcal{A}$ accepte avec une probabilité supérieure à 1-α
- si $\Delta(o, \mathcal{P}) > k$, alors $\mathcal{A}$ rejette avec une probabilité supérieure à β

La fig. 5.1 donne une interprétation intuitive de cette définition. Le point essentiel est que l'algorithme fournit *toujours* une réponse, acceptation ou rejet. Dans certains cas, la pertinence de cette réponse peut être quantifiée : le *risque de première espèce* (*fausse alarme, faux positif*) α est la probabilité de rejeter un objet correct ; le *risque de seconde espèce* (*faux négatif*) β est la probabilité d'accepter un objet fortement incorrect ; dans les deux cas, la probabilité porte sur la randomisation interne de l'algorithme, et non sur l'objet o d'entrée, qui est fixé. Le paramètre k mesure la région d'incertitude : lorsque l'objet n'est que faiblement incorrect, la pertinence de la réponse de l'algorithme de décision ne peut être quantifiée.

Les travaux dans les domaines d'informatique théorique connus comme *result checking* et *property testing* ont défini des vérificateurs pour des propriétés spécifiques à certains problèmes. Plus spécifiquement, un vérificateur est une procédure de décision sur la propriété ; la complexité de cette procédure est significativement plus faible que celle de la meilleure procédure de décision déterministe connue. Un exemple particulièrement simple est fourni par Blum [24, 25] dans son article fondateur. Soit un programme qui prétend trier un tableau. Pour chaque paire (vecteur d'entrées, vecteur de sorties) du programme, on peut vérifier, avec une complexité significativement plus simple que celle d'un programme de tri, que le vecteur de sortie est effectivement le vecteur d'entrée trié. Il faut en effet vérifier, d'une part que le vecteur de sortie est monotone, ce qui est linéaire en la taille du vecteur ; d'autre part, que les deux vecteurs sont identiques à une permutation près, ce qui peut être réalisé avec une probabilité arbitrairement proche de 1 par l'application itérative d'une fonction de hachage. Dans ce cas très élémentaire, l'algorithme

n'est qu'à demi probabiliste : une réponse négative est toujours exacte (ie la sortie n'est pas un tri de l'entrée, ou encore, le risque de première espèce peut être exactement nul) ; de même, la notion de région d'incertitude n'existe pas ($k = 0$).

Du même champ de recherche relève le *propety testing* [100], qui a défini des vérificateurs pour, par exemple, la linéarité d'une fonction, ou des propriétés structurelles de graphes (k-colorabilité) ; pour ces applications, les algorithmes présentent en général les trois types de risque (les trois paramètres α, β, k, sont strictement positifs). La principale différence entre le result-checking et le property testing est le recours du second à un oracle, qui fournit une réponse sur un problème de même nature que le problème initial, mais plus simple, alors qu'en result-checking, le vérificateur définit un algorithme associé à une propriété caractéristique de l'objet.

L'autre grande catégorie de procédure de décision est le *test d'hypothèses.* Il existe évidemment une immense littérature mathématique sur le sujet ; dans le cas le plus simple, on veut décider pour une loi statistique P_θ de paramètre inconnu θ dans $\mathbb{R}$, si $\theta \leq \theta_0$ ou bien $\theta > \theta_1$. La procédure consiste à définir une région d'acceptation H_0 (hypothèse nulle) de $\mathbb{R}^n$, tirer un échantillon empirique $(x_1, \ldots, x_n)$ de la loi, et décider de l'acceptation si et seulement $(x_1, \ldots, x_n) \in H_0$. Avec les notations précédentes, $\mathcal{E} = \mathbb{R}^n$, $\mathcal{P} = H_0$ et $k = \theta_1 - \theta_0$.

5.2.2 Les applications

Les algorithmes de result-checking et de property testing fournissent un ensemble de procédure de test qui peuvent être appliquées, soit aux jobs individuels, soit à des groupes de jobs qui réalisent un calcul commun, par exemple un tri parallèle. Les domaines d'application du Calcul Global les plus évidents ne relèvent cependant pas de cette approche : *ni les jobs, ni les ensembles de jobs, ne présentent des propriétés caractéristiques vérifiables.*

Les méthodes Monte-Carlo Elles visent à construire un échantillon d'une loi statistique. Dans de nombreuses applications de ce type utilisées en physique et en biologie, le programme simule un système complexe, dont seules les lois d'interaction locales sont connues. Les lois globales du système ne sont pas connues, contrairement aux applications classiques de calcul hautes performances où les équations globales sont disponibles. Il n'existe donc presque jamais de propriété caractéristique

positive, qui permette de valider individuellement un job. Suivant les applications, il peut en revanche exister des propriétés qui permettent d'infirmer les jobs.

La recherche d'évènements rares (SETI, distributed.net) La fréquence des jobs à résultat affirmatif ("on a trouvé un signal extraterrestre") est si faible que la vérification par réexécutions multiples est sans problème. Mais la masse des affirmations de non-intérêt d'un signal, ne peut être vérifiée que par réexécution.

Des applications typiquement multiparamètres définissent donc des jobs pour lesquels le seul test de vérification individuelle applicable à tous ou à la très grande majorité des jobs est la réexécution. Quel que soit le schéma de vérification (vote, m-premiers votants), la pénalité d'une vérification exhaustive est au minimum un facteur 2, et croît en puissance.

On peut alors chercher une solution dans la voie du test d'hypothèses. On va chercher à garantir *une borne supérieure du taux de défauts sur un ensemble de jobs.*

L'avantage de cette approche, de type contrôle de qualité, est sa généricité : comme elle ne dépend pas de la sémantique des jobs, elle s'implémentera entièrement dans le système de Calcul Global ; à l'inverse, même pour des applications pour lesquelles il existe des propriétés caractéristiques, la définition de l'algorithme de vérification qui teste ces propriétés est nécessairement à la charge du client, ce qui paraît dissuasif pour l'utilisation d'un système de Calcul Global. La généricité a cependant des limites qui dépendent du degré de tolérance aux fautes des applications. En reprenant les deux exemples précédents, on voit que la situation est très différente.

Les méthodes Monte-Carlo Les résultats de la simulation sont exploités par des procédures statistiques, pour estimer les paramètres ou la forme de la distribution (loi de puissance, lois plus complexes...). La vérification doit donc produire deux types de garanties : à la fois une borne sur le nombre d'erreurs, et sur l'amplitude des erreurs. Le niveau de demande sur ces garanties est spécifique de l'application. Par exemples, les méthodes statistiques *robustes* visent à tolérer des valeurs aberrantes d'amplitude inconnue, mais en nombre connu (par exemple, la médiane est un estimateur plus robuste de l'espérance que la moyenne).

La recherche d'évènements rares Essentiellement, ce type d'application n'est pas tolérant aux fautes. La vérification doit alors produire une garantie probabiliste très forte de correction

Dans le premier cas, et plus généralement dans tout cas où une tolérance aux fautes non nulle peut être quantifiée, on pourra définir des stratégies de test applicables à un environnement complètement pair à pair. A ce cas appartiennent aussi des applications de construction d'images (lancer de rayons par exemple), qui sont évidemment robustes : quelques erreurs locales ne pénalisent pas la qualité du résultat global. Dans le second cas, la définition de stratégies de test efficaces sera restreinte à des environnements où les collaborateurs sont identifiables, donc plus du type Calcul Global.

Les exemples présentés ci-dessus correspondent à l'utilisation la plus évidente des systèmes de calcul global. Des travaux récents [116] ont considéré un contexte très différent, celui d'applications distribuées, où les jobs sont le nœuds d'un graphe de tâches. Plus généralement, dans le contexte d'algorithmes distribués tolérants aux défaillances, soit par évitement, soit par auto-stabilisation, la connaissance d'une borne supérieure sur le taux de fautes peut être exploitée pour donner des garanties sur le comportement de l'algorithme.

5.2.3 Modélisation

La discussion ci-dessus conduit à définir l'unité vérifiable, appelée *batch* dans la suite, comme un ensemble de jobs. Un algorithme de vérification accepte ou rejette un batch dans son ensemble, ce qui a en particulier pour conséquence que tous les jobs d'un batch doivent attendre l'exécution de l'algorithme avant qu'aucun d'entre eux puisse être délivré au client.

La tolérance aux fautes de l'application est définie par deux paramètres : p_a est le taux de défauts acceptable sur un batch, et ϵ est le risque acceptable de faux négatifs, c'est à dire le risque que l'utilisateur accepte de prendre en utilisant un système qui ne garantit ses résultats que par un algorithme probabiliste.

Un échantillon de taille n d'un batch peut être modélisé comme un n-uplet de variables aléatoires binaires $(x_1, \ldots, x_n)$, avec $x_i = 0$ (resp. 1) signifiant que le job i est correct (resp. faux). En supposant les jobs indépendants, avec une probabilité p d'être faux, $(x_1, \ldots, x_n)$ suit une loi binomiale

$$P_p(x_1, \ldots, x_n) = p^{S_n}(1-p)^{n-S_n},$$

où $S_n = \sum_{i=1}^{i=n} x_i$.

L'hypothèse d'indépendance des jobs ne doit pas être confondue avec celle d'indépendance des saboteurs : les algorithmes de vérification échantillonnent

le batch pour évaluer p ; si les échantillons sont choisis uniformément dans le batch, le modèle des jobs indépendants est toujours applicable (à condition que la taille de l'échantillon soit petite devant celle du batch, ce qui est une contrainte évidente par ailleurs pour tout algorithme de vérification raisonnable). Un contre-exemple serait un test qui choisirait comme échantillon les n premiers jobs délivrés par le système, alors que les saboteurs sont plus rapides que les collaborateurs honnêtes. *Le modèle des jobs indépendants peut donc s'accommoder d'adversaires byzantins, en imposant une contrainte sur le processus d'échantillonnage.*

On veut donc définir un algorithme qui décide si le taux d'erreur d'un batch p est supérieur à p_a, avec un risque ϵ de faux négatif. L'algorithme a accès à un oracle, qui dit avec certitude si le résultat d'un job est un défaut. Ici et dans la suite un *défaut* est un job inexact et une *erreur* est une décision erronée de l'algorithme de vérification. L'oracle sera typiquement réalisé par réexécution sur des machines fiables, ou par vote.

Dans le cas du test de propriété (ou de test de résultat), l'algorithme de test suffit à définir le processus de vérification, et sa performance peut être définie par l'inverse du nombre d'appels à l'oracle. Dans le cas du Calcul Global, l'algorithme de vérification est plus complexe. Par exemple, lorsque l'algorithme doit rejeter un batch, tous les jobs exécutés l'ont été inutilement, puisqu'un batch se délivre en totalité ou pas du tout ; au coût du ou des tests s'ajoute un coût d'exécution des jobs. L'algorithme de vérification inclut donc, outre un ou des algorithmes de test, la consommation des ressources d'exécution, et la méthode d'échantillonnage.

La performance d'un algorithme de vérification, pour p_a et ϵ donnés, comprend deux composantes : la consommation de ressources et la pénalité de correction. Pour modéliser la consommation de ressources, soient B la taille du batch, $s(p)$ le nombre d'appels à l'oracle et $r(p)$ le nombre de jobs lancés avant un rejet, c'est à dire les ressources consommées inutilement ; on suppose que, lorsque l'algorithme accepte, il ne lance qu'une fois les jobs du batch. Le surcoût en ressources d'un algorithme de vérification, pour une valeur p de la probabilité de défaut, est alors $s(p)$, dans tous les cas (que l'algorithme accepte ou rejette), plus $r(p)$ lorsque l'algorithme rejette, le tout étant normalisé par la taille B du batch :

$$C(p) = \frac{1}{B}\left(s(p) + r(p)P_p(\text{rejet})\right), \tag{5.1}$$

Ce critère de performance a un défaut : il n'est pas vrai en général que le

surcoût soit monotone par rapport à la probabilité de défauts sous-jacente p. Dans le contexte du test d'hypothèse, pour certaines lois de probabilités usuelles et des tests simples non adaptatifs, cette propriété est souvent vraie. Pour des tests adaptatifs, la propriété de monotonie peut être fausse ; c'est le cas des algorithmes que nous proposons dans la suite.

La définition du surcoût doit alors faire intervenir la distribution statistique du taux de défauts. Si Q est la loi de p, on peut envisager l'indicateur de surcoût suivant :

$$\mathcal{C}_a = \frac{1}{B} E_Q[C(p)|p \leq p_a] = \frac{1}{BQ([0,p_a])} \int_0^{p_a} C(p)\mathrm{d}Q, \tag{5.2}$$

qui est la moyenne (sous Q) du surcoût lorsque $p \leq p_a$, donc lorsque l'algorithme doit accepter.

Il faut souligner que cet indicateur mesure le *surcoût* dans un état *stable*. Par exemple, pour un système sans sabotage, $Q(p=0)=1$; tout test raisonnable doit accepter systématiquement un batch parfait, *i.e.* $P_p(\text{rejet}) = 0$; donc $\mathcal{C}_a = s(0)$: le surcoût est exactement le nombre d'appels à l'oracle.

Par état stable, on entend un état où le système est en état de servir ses consommateurs, *i.e.* $p < p_a$. Pour un système qui facturerait ses services, $\mathcal{C}_a$ représenterait le surcoût qui doit être compensé et qui peut l'être, puisque le système doit parvenir à délivrer des résultats : pour un surcoût attendu de 0,5, le consommateur doit fournir 1,5 fois les ressources correspondant à ses jobs, et doit recevoir les résultats. Il serait plus malaisé de donner un sens à une fonction de surcoût global : lorsque le système n'est pas en état de fonctionner, le système ne peut rien délivrer ; la seule notion de coût qui ait un sens est le nombre d'appels à l'oracle, qui définit la quantité de ressources gaspillées avant d'interrompre le fonctionnement pour le corriger si possible par des actions externes au test.

La distribution Q sur $[0,1]$ modélise le comportement des saboteurs.

- Q concentrée sur 0 et 1 : $Q(0) = q, Q(1) = 1-q$, qui correspond à une alternance de phases d'attaques massives et coordonnées (tous les jobs faux, $p=1$) et de situation idéale ($p=0$).
- Q en loi de puissance : $Q_\lambda(p \leq x) = x^\lambda; 0 < \lambda \leq 1; x \in [0,1]$, décrit une distribution dont la concentration autour du taux de défauts nul dépend du paramètre λ.
- Q uniforme : $Q(p \leq x) = x$. C'est le cas précédent avec $\lambda = 1$.

5.3 Algorithmes adaptatifs

Le contexte du Calcul Global ou P2P permet de faire l'hypothèse que le fonctionnement du système présente deux comportements distincts.

Fonctionnement normal Il comprend deux situations :
- La plupart des collaborateurs ne sabotent pas. C'est par exemple le cas si le système peut contrôler l'origine des jobs (cf section 5.4), ou si le sabotage est rendu difficile (code binaire, cryptage de code).
- Les saboteurs falsifient systématiquement leurs résultats, volontairement, comme dans l'exemple de SETI@home, ou par erreur d'installation.

Attaque subtile Dans ce scénario, un grand nombre de collaborateurs sont en réalité des saboteurs. Si ces collaborateurs sabotaient systématiquement (cas précédent), le système doit âtre capable de les éliminer ; il peut continuer à fonctionner avec les collaborateurs sérieux. L'attaque la plus subtile consiste donc à fournir des collaborateurs qui falsifient seulement une fraction des résultats, et qui sont ainsi difficiles à identifier.

La possibilité de scénarios très différents milite pour des algorithmes de vérification *adaptatifs* qui sont optimisés pour la situation la plus fréquente. La réalisation d'une attaque subtile est complexe : l'entité responsable du sabotage doit contrôler une fraction significative de collaborateurs, en organiser l'anonymat, et falsifier non systématiquement. L'algorithme de test doit donc à la fois être optimisé pour traiter le cas du fonctionnement normal et être capable de détecter une situation d'attaque subtile.

5.3.1 Le test séquentiel de Wald

Un résultat standard de statistique (théorème de Neyman-Pearson) montre qu'il existe un test optimal parmi les tests non adaptatifs, c'est à dire ceux qui prélèvent un échantillon de taille fixe, pour des distributions de défauts binomiales ou plus généralement pour toutes les distributions de taille exponentielles. Le principe, qui était connu bien avant la preuve d'optimalité, est de définir la région d'acceptation par le nombre total de défauts dans l'échantillon :

$$H_0 = \{(x_1, \ldots x_n)\} | S_n \leq c\}.$$

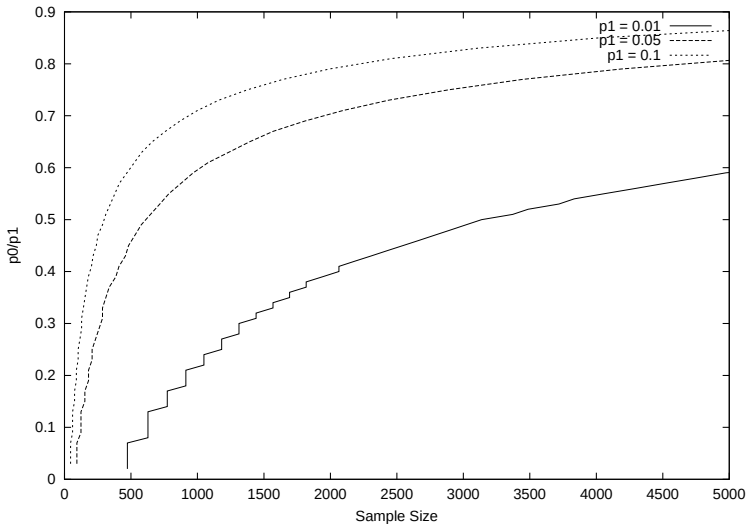


FIG. 5.2 – Test non adaptatif - Loi binomiale

c et n, qui instancient le test, sont déterminés par

$$P_{p_0}(H_0) \geq 1 - \alpha \text{ et } P_{p_1}(H_0) < \beta.$$

Ce test est proposé dans tous les environnements statistiques, du tableur aux environnements hautement spécialisés. Son coût est très élevé, comme le montre la fig. 5.2 : pour atteindre une précision relativement faible ($p_0 = p_1/2$), la taille de l'échantillon est de l'ordre de 100 pour $p_1 = 0.1$, mais de plusieurs milliers pour $p_1 = 0.01$.

Le test séquentiel défini par Wald [193, 175] formalise l'idée intuitive qu'un test partiel est suffisant si les données antérieures impliquent l'acceptation ou le rejet. La fig. 5.3 (a) décrit le fonctionnement du test : le test commence avec un échantillon de taille 1 ; à l'étape m, si le nombre de défauts dans l'échantillon $(x_1, \ldots, x_m)$ est supérieur (resp. inférieur) à une valeur qui dépend de m, le test rejette (resp. accepte) ; si le nombre de défauts est compris entre ces deux valeurs, la taille de l'échantillon est augmentée. La taille de l'échantillon devient alors une variable aléatoire, et une première approximation du coût du test est la moyenne de cette variable pour la probabilité P_p, qu'on note dans la suite $E_p(n)$. On montre [193] que le test termine ($E_p(n) < +\infty$). Le choix du test séquentiel a deux avantages sur le test statique (fig. 5.3 (b)) :

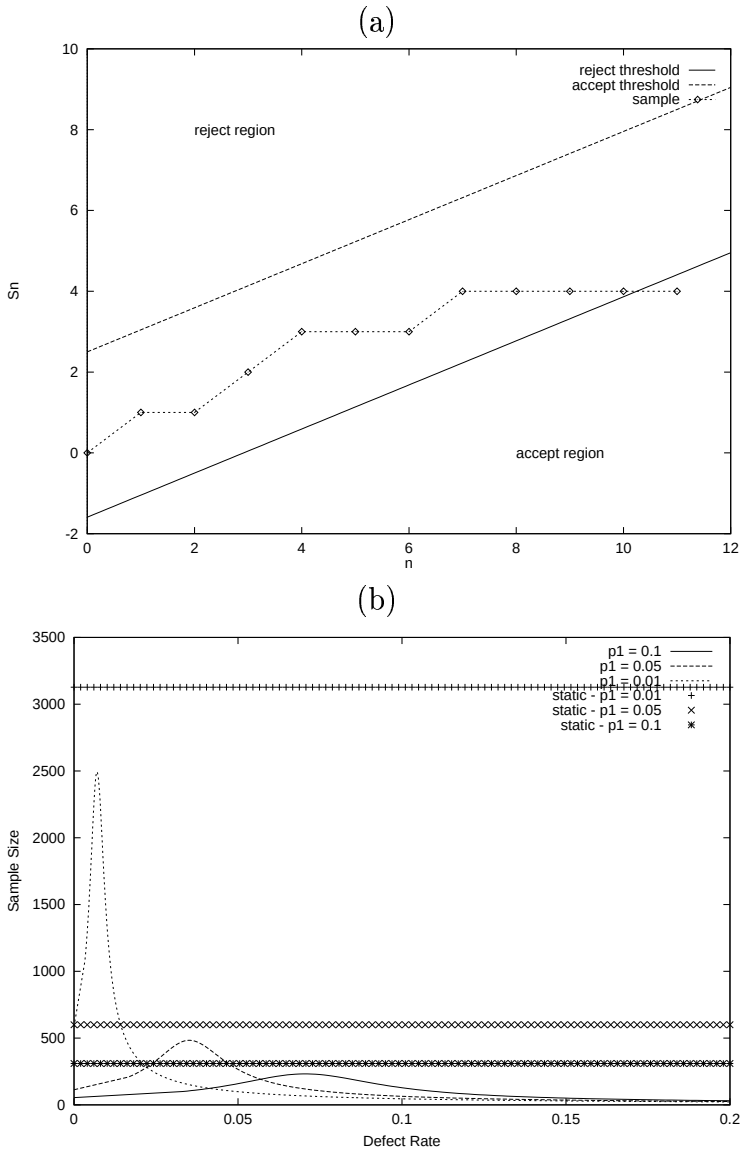


FIG. 5.3 – Le test séquentiel. (a) Principe. (b) Taille de l'échantillon pour le test séquentiel et le test statique (loi binomiale). Pour tous les exemples, $p_0 = p_1/2$.

- En moyenne, son coût est toujours (quel que soit le taux de défauts) moins élevé que celui du test statique.
- Il est adaptatif; le coût croît jusqu'à un maximum entre p_0 et p_1, et décroît rapidement ensuite. Le gain par rapport au test statique est un facteur 5 pour la situation de fonctionnement normal stable, où les collaborateurs ne sabotent pas, et de plusieurs ordres de grandeur pour un sabotage massif.

Dans la suite, on note $L(p)$ la probabilité que le test accepte (toujours pour la loi P_p). Par construction, $L(0) = 1, L(p_0) = 1 - \alpha, L(p_1) = \beta$ et $L(1) = 0$.

On rappelle les résultats de [193].

$$L(p) = \frac{(\frac{1-\beta}{\alpha})^h - 1}{(\frac{1-\beta}{\alpha})^h - (\frac{\beta}{1-\alpha})^h}$$

où h est défini par

$$p = \frac{1 - (\frac{1-p_1}{1-p_0})^h}{(\frac{p_1}{p_0})^h - (\frac{1-p_1}{1-p_0})^h}$$

Les valeurs de h correspondant respectivement à $p = 0, p_0, p_1, 1$ sont $+\infty, 1, -1$ et $-\infty$

De plus

$$E_p(n) = \frac{L(p)logb + (1 - L(p))loga}{plog\frac{p_1}{p_0} + (1-p)log\frac{1-p_1}{1-p_0}}$$

avec $a = \frac{1-\beta}{\alpha}$ et $b = \frac{\beta}{1-\alpha}$

Les tests séquentiels sont massivement utilisés dans de nombreux domaines. Dans le domaine médical (essais thérapeutiques, enquêtes épidémiologiques), l'essai doit se terminer dès que des données significatives sont disponibles, pour des raisons éthiques (lorsqu'une des traitements est clairement meilleur que l'autre) ou économiques, ce qui motive une littérature considérable portant sur des variantes du test (pour une bibliographie et une analyse détaillée, on pourra consulter [88]). En contrôle de processus industriel, le test non-adaptatif correspond aux V-tests, et le test séquentiel aux procédures CUSUM. [175] note la relation entre ce type de test et la programmation dynamique. Enfin, le concept de *crédibilité* développé par [166] cherche à exprimer la même idée, mais avec des outils mathématiques trop élémentaires.

5.3.2 Algorithmes de vérification

Le cas le plus difficile est celui où les collaborateurs ne sont pas identifiables : la composition des batchs ne peut pas être améliorée par élimination des collaborateurs suspects.

Test séquentiel simple

Il s'agit de l'application directe du test de Wald. Pour un taux d'erreur p, le coût d'échantillonnage $s(p)$ est égal à $E_p(n)$, et la probabilité de rejet est simplement $1 - L(p)$. Pour être conforme aux spécifications de l'utilisateur, il faut choisir $p_1 = p_a$ et $\beta = \epsilon$. D'après (l'équation 5.2),

$$\mathcal{C}_a = \frac{1}{B} E_Q[E_p(n)|p \leq p_a] + E_Q[1 - L(p)|p \leq p_a]$$

La spécification complète de l'algorithme demande de définir p_0, α et la taille du batch B.

Dans l'expression précédente, le terme $E_p(n)$ croît avec p_0, pour p fixé, et le second terme décroît : un test est d'autant plus sélectif, en acceptation, que p_0 est plus proche de p_1, et demande donc un échantillon de plus grande taille. Ces inégalités passent aux espérances en Q.

On ne peut pas obtenir de formules explicites pour $\mathcal{C}_a$ à partir des expressions de 5.3.1. La fig. 5.4 présente la fonction de surcoût obtenue par intégration numérique, en fonction de p_0, et pour les distributions de probabilités sur p décrites en 5.2.3. On ne considère que les valeurs de p_0 telles que $B \leq E_{\max}(n)$. Rappelons que $\mathcal{C}_a$ décrit le surcoût, c'est à dire la consommation de ressources non directement productives.

Le résultat le plus évident est que le test simple fournit des performances acceptables en fonctionnement normal, lorsque la distribution de p est très concentrée en 0. C'est le cas, soit lorsque la très grande majorité des collaborateurs ne sabotent pas, soit dans le cas des attaques massives, qui sont détectées et éliminées. En revanche, le test simple devient très coûteux en situation d'attaque permanente, modélisée par une distribution uniforme : le sabotage consiste alors à fournir assez de résultats faux pour demander une consommation de ressources excessive, et assez de résultats justes pour que le système ne réagisse pas efficacement. Le test simple est donc très vulnérable à une attaque en déni de service.

La définition d'une valeur précise pour p_0 dépend de l'importance relative des critères de taux de faut rejet d'une part, et de surcoût d'autre part. Une

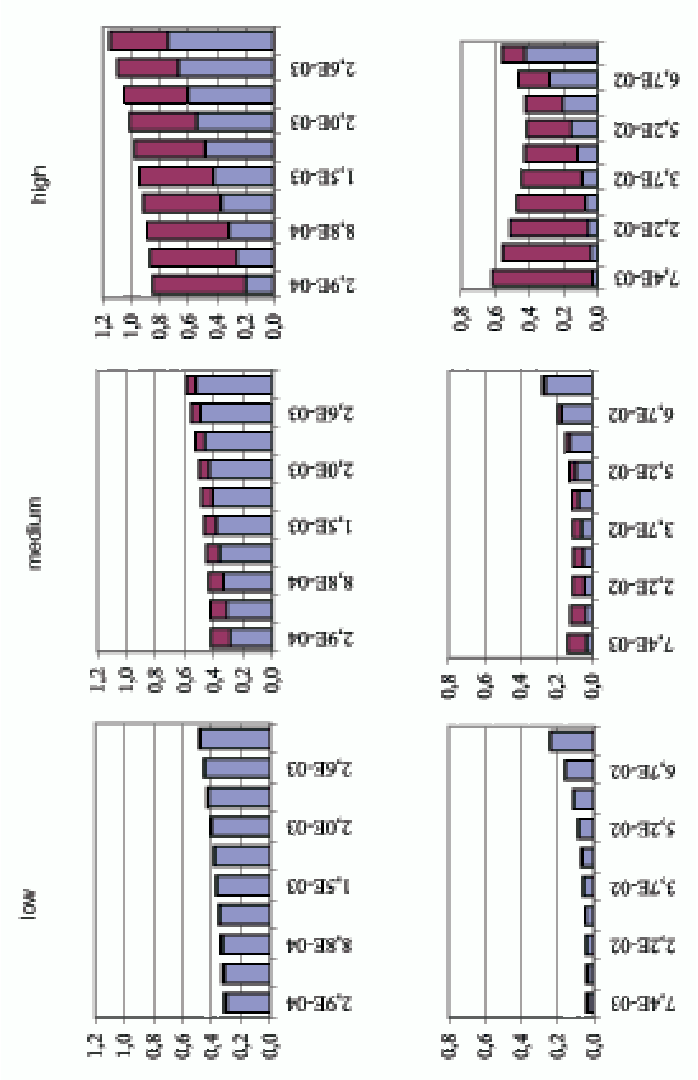


FIG. 5.4 – Surcoût du test simple en fonction de p_0. Pour chaque histogramme, la partie supérieure est le surcoût d'erreur et la partie inférieure le surcoût d'appel à l'oracle. $\alpha = \beta = 5\%$. La colonne *low* correspond à $\lambda = 0.01$ (distribution des défauts concentrée en 0) ; la colonne *medium* à $\lambda = 0.1$; la colonne high à $\lambda = 1$ (distribution uniforme). La ligne supérieure correspond à $p_1 = p_a = 0.01$, la ligne inférieure à $p_1 = p_a = 0.1$. $B = 1000$

analyse plus détaillée montre que, lorsque p_0 est grand, le surcoût élevé est dû à la taille de l'échantillon, avec peu de faux rejets. Le second critère de performance, le taux de faux rejet, est donc satisfait. En revanche, lorsque p_0 est petit et que la distribution du taux de défauts n'est pas très concentrée en 0, les performances sont mauvaises pour les deux critères : le surcoût reste élevé et taux de faux rejets aussi.

Le domaine des tests statistiques étant largement exploré, on ne peut pas espérer améliorer le taux de faux rejets. En revanche, la définition d'une procédure différente de mise en oeuvre du test peut parvenir à diminuer le surcoût.

Test deux phases

Le détection précoce d'un rejet diminue le surcoût, puisque le batch n'est pas exécuté en totalité. Un test 2 phases commence par évaluer la réaction du système à un batch particulier, en testant tous les jobs lancés. Lorsqu'une décision est prise, si elle est positive, le reste de jobs du batch est lancé, puis un nouveau test simple par sélection aléatoire des résultats de l'ensemble du batch est effectué. Le batch n'est accepté que si les deux tests acceptent. La probabilité d'acceptation, sous un taux de défauts p est alors $L^2(p)$, et on vérifie facilement que

$$C^{\text{2phases}}(p) = \frac{1}{B}[(1 + L(p))E_p(n) + L(p)(1 - L(p))].$$

Le deuxième test est nécessaire, les premiers jobs lancés n'étant pas représentatifs du système, sauf si les adversaires sont indépendants. Une hypothèse beaucoup plus faible que celle d'adversaires indépendants est de supposer que le taux de défauts reste constant entre la première et la seconde phase, ce qui équivaut à supposer que l'adversaire ne peut pas déceler à quelle fréquence les jobs sont testés. Sous cette hypothèse, les valeurs de α et β de ce nouveau test doivent être adaptées, avec en particulier $\beta = \sqrt{\epsilon}$, pour obtenir la même sensibilité globale.

Lorsqu'il n'y a pas de défauts, en utilisant la valeur adaptée de β, les formules de 5.3.1 donnent $E_p^{\text{2phases}}(n) = E_p(n)/2$; comme $L(0) = 1$, on a $C^{\text{2phases}}(0) = C(0)$, donc le test 2 phases n'est pas plus coûteux que le test simple. Dès que le taux de défauts n'est plus nul, la comparaison présentée fig. 5.5 montre que le test en deux phases diminue considérablement le surcoût pour une large gamme de valeurs de p_0. L'adaptation de la valeur de β permet

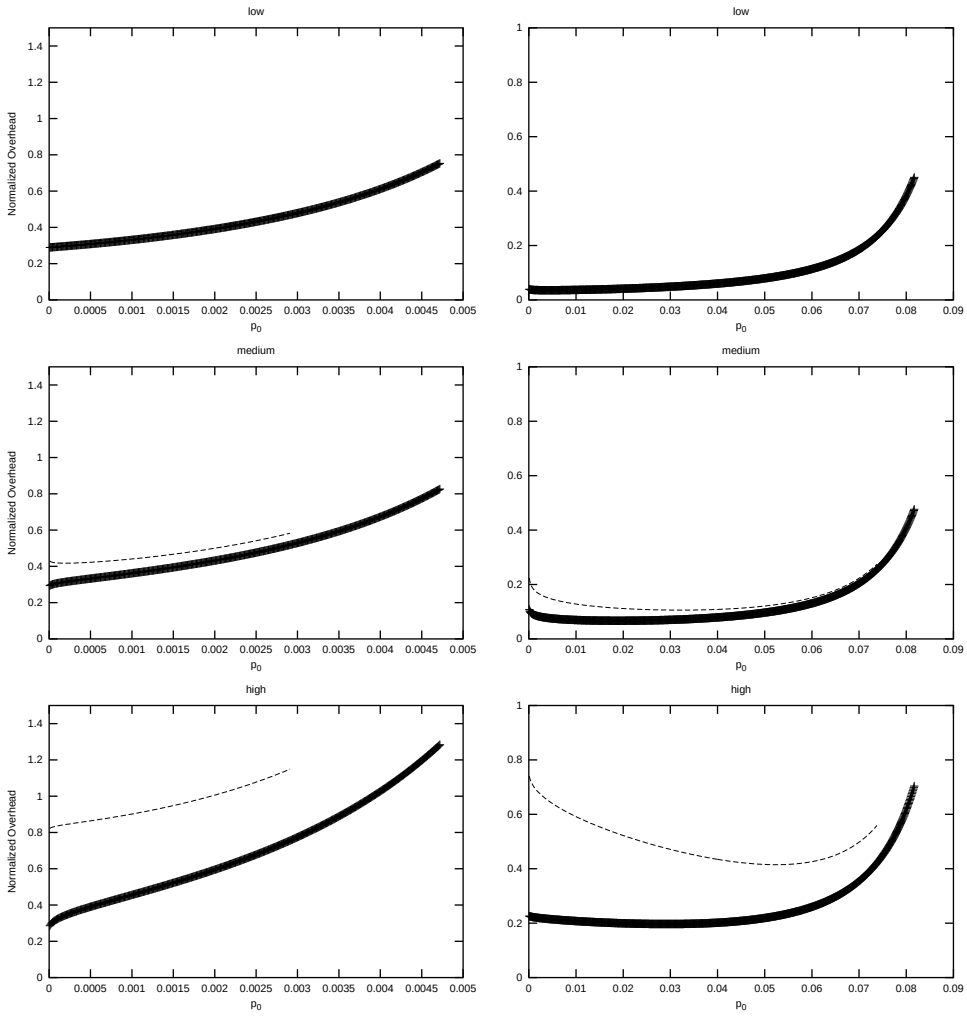


FIG. 5.5 – Comparaison des surcoûts test simple et test 2-phases (en gras). La colonne de gauche correspond à $p_1 = p_a = 0.01$; la colonne de droite à $p_1 = p_a = 0.1$. $\alpha = \beta = 5\%$. Medium, low and high comme dans la fig. 5.4

également d'atteindre des valeurs de p_0 plus grandes que celles du test simple mais avec un surcoût supérieur). Le test 2 phase a donc deux avantages.

- Il rend le système beaucoup plus robuste en face d'une situation d'attaque.
- Il élargit considérablement la gamme de choix pour le compromis entre les coûts de rejet et le surcoût de consommation de ressources.

Si l'hypothèse d'un taux de défauts constant entre les deux phases n'est pas valide, donc qu'on suppose un adversaire omniscient, la situation est plus contrastée [p34]. Le test simple est meilleur lorsque le taux de défaut est concentré sur 0 ; dans le cas où p est exactement nul, le surcoût du test 2 phases est double de celui du test simple ; si le taux de défauts est uniforme, le test 2 phases est meilleur pour les petites valeurs de p_0, comme dans le cas précédent.

La taille du batch B est motivée par un ensemble de contraintes externes au test lui-même, en particulier cohérence –un batch doit être composé de jobs correspondant au même code, pour que le test ait un sens– et réalisme –quelques dizaines à un millier de jobs–. On peut cependant considérer deux stratégies : taille fixe, ou adaptée aux paramètres du test. Une adaptation à priori intéressante est

$$B = E_{\max}(n) = \max\{E_p(n) | p \in [0,1]\}).$$

5.4 Crédibilité des collaborateurs

Dans un cas plus simple, l'association entre jobs et collaborateurs est identifiable. C'est le cas par exemple si un certificat est exigé des collaborateurs, et est associé à chaque résultat. Il est alors possible de tester non seulement les batchs, mais les collaborateurs eux-mêmes, et d'éliminer les jobs provenant de collaborateurs suspects. Cette élimination doit rester progressive : d'une part, une erreur isolée peut être transitoire (fausse manipulation) ; d'autre part, il est possible d'exploiter les résultats d'un saboteur non systématique, comme on l'a vu dans le cas précédent.

Pour ce test, la contrainte pertinente est d'accepter (avec une forte probabilité $1-\alpha$) toute production correcte, donc on aura $p_0 = p_a$. La définition des autres paramètres (β et p_1) sera fournie par des contraintes extérieures, par exemple le rapport entre la taille du test et le nombre de jobs confiés au collaborateur. La taille du sous-batch étant limitée, le test séquentiel simple

est probablement le seul applicable. Le résultat du test est donc, d'une part l'acceptation ou le rejet du sous-batch confié au collaborateur, d'autre part une évaluation du collaborateur, qui peut être binaire, fiable ou suspect. Ces deux résultats peuvent être exploités utilement.

Au niveau de l'évaluation d'un batch, l'effet du test est d'améliorer la qualité du batch obtenu. Lorsqu'un batch est complètement disponible, les résultats qui le composent ne proviennent que des collaborateurs restants, non suspects. Le batch doit alors être re-testé avec $p_1 = p_a$, pour garantir la conformité avec les tolérances initiales ; comme le batch est très probablement d'excellente qualité, une valeur de rejet p_0 très basse est admissible, ce qui permet un test rapide sans augmenter trop significativement le risque d'erreur.

L'utilisation du test pour l'évaluation des collaborateurs est discutée dans la section suivante.

5.5 Fiabilité globale

Les algorithmes de test doivent être intégrés dans un processus plus général, qu'on appellera le *contrôleur*. L'objectif général du contrôleur est d'adapter l'affectation des ressources du système de calcul global en fonction des critères de qualité définis extérieurement et du comportement du système. Une modélisation quantitative du fonctionnement du contrôleur demanderait une base d'expérience sur le comportement du système, en particulier les scénarios d'attaque, qui n'existe pas actuellement ; on se bornera à quelques idées très simples.

Un système de calcul global contiendra trois types de ressources : les collaborateurs absolument fiables, qui sont sous le contrôle direct de l'administration du système, les collaborateurs identifiés, et les autres, les deux dernières pouvant être volatiles.

Les collaborateurs absolument fiables ont pour rôle essentiel de réaliser les tests individuels de résultats. Leur nombre ne pouvant être que très limité, ils constituent le goulet d'étranglement du système. L'extension du pool de collaborateurs de test aux collaborateurs identifiés est un sous-produit du test de batch. Au cours du test de batch, les collaborateurs identifiables sont testés en permanence. On peut alors définir un seuil sur le nombre de classifications du collaborateur comme non-suspect au-delà duquel il est intégré dans le pool. La définition du seuil est elle-même de nature statistique,

mais plutôt du type test d'évènements rares. Le collaborateur promu fiable doit cependant continuer à être contre-testé par les ressources absolument fiables.

Un problème beaucoup plus difficile est la gestion des situations de rejet. Lorsque le taux de défauts est estimé dépasser le seuil admissible, laisser fonctionner le système sans intervention particulière signifierait simplement chercher à réaliser l'évènement statistiquement improbable de faux négatif.

Il est particulièrement important que le système de calcul global puisse contraindre les collaborateurs à recharger le code des applications utilisateur. Bien évidemment, si le code de l'application de calcul global lui-même n'est pas protégé, cette contrainte ne peut pas être complètement vérifiée. Pour les collaborateurs identifiés, le rechargement peut être sélectif, sur identification d'un collaborateur comme saboteur. Pour les collaborateurs non identifiés, le rechargement global représente une pénalité importante en charge réseau, avec le risque d'effets dissuasifs d'engorgement du service ; on peut alors envisager des rechargements aléatoires, ou à partir des identités supposées.

5.6 Une étude de cas

Un exemple intéressant, car probablement généralisable, est celui des Monte-Carlo utilisés dans l'expérience Auger.

5.6.1 L'observatoire Auger

L'expérience Auger étudie les rayons cosmiques d'ultra-haute énergie. Les rayons cosmiques sont des particules électromagnétiques à des énergies supérieures à 10^{19} eV (par comparaison, les accélérateurs atteignent au mieux 10^{12}eV). Ils sont aussi extrêmement rares : à 10^{19} eV, un évènement par km^2 et par an ; à 10^{20} eV, un évènement par km^2 et par siècle. L'origine de ces rayons cosmiques constitue un problème essentiel de physique fondamentale : dans le modèle actuel de l'univers, il n'existe pas de mécanisme qui permette à ces particules d'atteindre l'atmosphère terrestre, en particulier à cause du freinage par le rayonnement de fond de l'univers. L'expérience Auger constitue une des expériences cruciales pour les modèles généraux de la physique.

Les rayons cosmiques peuvent être détectés au niveau terrestre par les *gerbes atmosphériques* qu'ils produisent. Dans la haute atmosphère, la collision entre la particule primaire et une molécule d'air entraîne une désintégration

Thinning relatif	10^{-4}	10^{-5}	10^{-6}	10^{-7}
Temps de calcul	1mn 43s	16mn 52s	2h 50mn	24h58mn
Stockage	541KB	5MB	46MB	589MB

TAB. 5.1 – Influence du thinning sur le coût de simulation - Gerbe à 10^{20} eV - Xeon 2,4GHz

et l'apparition de nouvelles particules qui partagent son énergie ; ces particules vont à leur tour entrer en collision avec des molécules d'air, et l'ensemble du processus constitue la gerbe. Une collaboration mondiale construit actuellement un immense appareillage de détection de ces évènements en Argentine [10]. Cet appareillage est composé d'un réseau de détecteurs de particules, sur une surface de 3000 km^2,qui échantillonnent en permanence le flux de particules électromagnétiques, et détectent ainsi une gerbe lorsqu'elle arrive au sol, et de détecteurs de fluorescence qui enregistrent le développement de la gerbe dans l'atmosphère.

L'observation des rayons cosmiques est très indirecte, comme dans la plupart des expériences modernes de physique. Les quantités intéressantes *in fine* pour la physique, en particulier la nature de la particule électromagnétique primaire, son énergie initiale et son angle d'incidence, doivent être reconstruits à partir des mesures des effets physiques produits dans les détecteurs par la gerbe. La définition des modèles de reconstruction repose largement sur la *simulation numérique* des gerbes et des détecteurs. Dans l'environnement actuellement utilisé par l'expérience Auger, la simulation des gerbes domine très largement les temps de calcul. On ne considèrera donc que ces simulations dans la suite.

5.6.2 La simulation des gerbes atmosphériques

Les simulations prennent en entrée les paramètres physiques, en particulier les caractéristiques de la particule primaire, et fournissent un échantillonnage du développement de la gerbe et des particules qui atteignent le sol. Les codes de simulation utilisés par la collaboration Auger sont Aires [170] et Corsika [105].

Le phénomène physique de la gerbe est un processus complexe et partiellement aléatoire, en particulier à cause de la faible densité de la haute atmosphère. En conséquence, la simulation est de type Monte-Carlo : la

physique de l'interaction des particules de la gerbe avec l'atmosphère est déterministe, mais la localisation de cette interaction est probabiliste.

Cette première source d'aléa provient de la modélisation de la réalité physique. Il s'y ajoute une source interne, qui provient des contraintes de la simulation numérique. Il est impossible de suivre toutes les particules, à la fois en temps de calcul et en encombrement : une particule primaire à 10^{20}eV crée une gerbe de l'ordre de 10^{11} particules. Dans le code Aires, les simulations sont rendues possibles par un algorithme d'échantillonnage statistique, le *thinning*, qui permet de propager seulement une fraction représentative de la gerbe. Le paramètre principal du thinning, l'énergie de seuil E_s, fournit un réglage précis de la consommation de ressources : le temps de calcul et le volume des résultats dépendent exponentiellement du thinning relatif, *i.e.* E_{primaire}/E_s, avec un facteur 8 à 10 suivant les modèles d'interaction (table 5.1). L'effet général d'une diminution du thinning est naturellement de diminuer la fluctuation statistique des résultats, mais de façon différenciée suivant les paramètres physiques mesurés.

5.6.3 Certification

Les sections précédentes ont présenté en détail les méthodes de test qui permettent de borner le taux de fautes. La spécificité du traitement de l'application de simulation des gerbes se situe donc au niveau de la vérification individuelle des résultats.

Principe

L'objectif est évidemment de diminuer le coût de la vérification par rapport à la réexécution pure et simple. La dépendance exponentielle en temps de calcul sur le thinning permet d'obtenir une approximation pour chaque gerbe G à tester, en calculant un échantillon de contrôle $\S(G)$ de contrôle de la même gerbe avec un thinning plus élevé, avec un facteur de coût contrôlable et faible. Il faut alors tester la compatibilité statistique du résultat à vérifier avec cet échantillon.

Le problème est alors de définir le test de compatibilité. Le test rigoureux de compatibilité d'un item avec un échantillon relève du contexte général du traitement des outliers en statistique. Un *outlier* est une valeur extrémale (plus grande ou plus petite) dans un échantillon d'une distribution ; le problème de la *détection des outliers*, qui est le seul concerné ici, consiste à déterminer

si cette valeurs est aberrante (contamination, fausse manœuvre etc.) ou légitime ; de très nombreuse études ont analysé formellement ou empiriquement la puissance de divers tests contre des alternatives à distribution de forme connue, et tabulé les seuils d'acceptation (une synthèse est présentée [14]).

Falsification des gerbes

Les données définissant les résultats d'une simulation de gerbes forment un volume à la fois important et extrêmement structuré. Une falsification directe des résultats semble donc très improbable ; le mode normal de falsification consiste à substituer les résultats de la simulation d'une gerbe $G1$ à ceux d'une autre gerbe $G2$.

Il faut donc vérifier, au vu des résultats, que les *paramètres d'entrée* d'un gerbe simulée sont ce qu'ils prétendent être. On pourrait évidemment représenter cette situation comme une fonction de l'espace des paramètres dans l'espace des résultats, à inverser, mais cette représentation n'est guère opératoire : la fonction n'a pas d'autre définition que la simulation elle -même. On pourrait aussi la représenter comme un problème de classification ; en première approximation, l'interprétation physique des paramètres d'entrée et de sortie permet de limiter considérablement le champ des critères pertinents.

A partir de l'ensemble des données résultats d'Aires, on peut obtenir par post-traitement un ensemble de paramètres de sortie, en temps négligeable. Ces paramètres sont beaucoup trop sommaires pour capturer l'ensemble des propriétés de la gerbe utiles à la physique, mais dépendent de façon complexe de l'ensemble des données résultats, donc constituent de bons candidats pour le test de falsification.

La difficulté pour la définition du test provient de ce que les distributions de probabilité de ces paramètres ne sont pas simples. La double source d'aléa, physique et de simulation, a pour conséquence que la distribution des résultats d'un ensemble de simulations pour un même jeu de paramètres d'entrée est la superposition d'un ensemble de lois différentes.

L'étude qui suit porte sur la vérification des paramètres énergie et nature de la particule.

Choix du test

Energie Les résultats les plus susceptibles d'être discriminants pour le paramètre d'entrée énergie sont ceux qui décrivent le maximum de la gerbe : $X_{\max}$ l'altitude du point où le nombre de particules chargées atteint son maximum et $N_{\max}$, la valeur du maximum. En effet, le développement de la gerbe est initialement croissant, chaque interaction produit de nouvelles particules, puis devient décroissant, lorsque les particules secondaires n'ont plus assez d'énergie pour que les interactions produisent de nouvelles particules (en deçà d'une certaine énergie appelée *énergie de coupure*, les particules ne produisent plus d'interaction et ne sont plus suivies). L'altitude $X_{\max}$ en elle-même n'est pas suffisamment discriminante, car l'effet de l'altitude de la première interaction introduit une variabilité importante. En revanche, le nombre de particule $N_{\max}$ ne dépend pas de l'altitude de première interaction. En outre, $X_{\max}$ et $N_{\max}$ sont obtenus par un fit des données de l'ensemble du développement de la gerbe avec une fonction (fonction de Glaisser-Hillas) ; le maximum de la gerbe est ainsi une signature du développement de la gerbe, dont on peut espérer qu'il caractérise les paramètres d'entrée de type énergie (énergie de la particule incidentes et énergies de coupure).

Nature Le maximum de la gerbe n'est pas suffisamment discriminant pour la nature de la particule. La simulation admet l'hypothèse dite d'indépendance, qui entraîne une sorte d'homothétie des gerbes. Un paramètre discriminant est donc à chercher dans des quantités absolues. Pour des raisons physiques, le nombre de muons au niveau du sol (*NMuons*) est le meilleur candidat.

Le test considéré est l'écart à la moyenne : $\frac{|x-m|}{s} < c$, où m et s sont respectivement des estimateurs de la moyenne et de l'écart type dans l'échantillon de contrôle. L'estimateur de m est la médiane, et celui de s l'estimateur sans biais. Dans le cas d'une distribution gaussienne, ce test peut être interprété comme un test de maximum de vraisemblance. Plus précisément, le test accepte si $(\frac{|x_1-m_1|}{s_1} < c_1) \wedge (\frac{|x_2-m_2|}{s_2} < c_2)$, où le premier test est relatif à $N_{\max}$ et le second à *NMuons*.

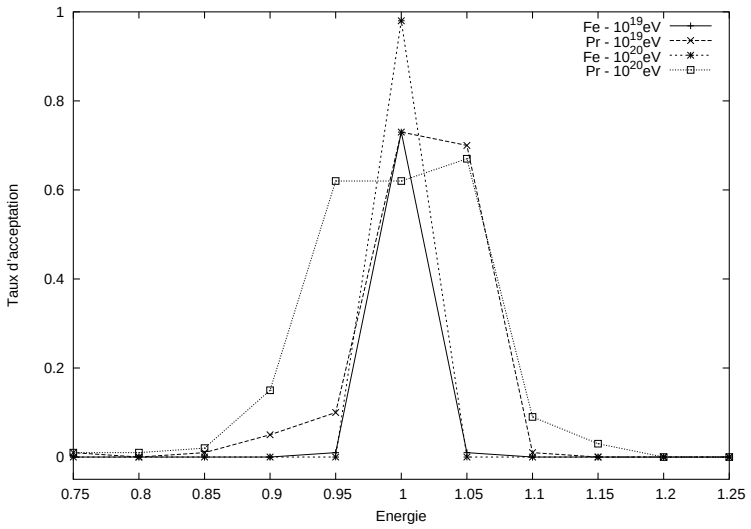


FIG. 5.6 – Performance du test - discrimination de l'énergie

Mesures

Une campagne de mesures a été menée en utilisant la base de données centrale de résultats d'Auger et l'installation XtremWeb du LAL pour l'acquisition de résultats. Les conditions de l'expérience sont les suivantes.

L'énergie des gerbes à vérifier varie de $0.75X$ à $1.25X$, avec $X = 10^{19}$eV ou $X = 10^{20}$eV, ce qui correspond au domaine d'énergie utile pour les simulations Auger. La particule initiale peut être soit Fer, soit Proton. Pour chaque gerbe G à vérifier , l'ensemble de contrôle $\S(G)$est composé de 20 gerbes, aux paramètres nominaux de G, sauf le thinning qui est 10^{-4}.

Les performances du test sont évaluées sur un lot de gerbes à $1,0X$, qui jouent le rôle des gerbes éventuellement falsifiées. Pour disposer d'un échantillon significatif de gerbes falsifiées, les gerbes disponibles dans la base de données centrale d'Auger ont été utilisées. Elles correspondent en général à un thinning de 10^{-6}, ce qui constitue un pire cas : les résultats de simulation sont d'autant plus dispersés que le thinning est élevé et 10^{-6} est la limite inférieure pour que les simulations détaillées aient un sens physique. La taille de l'échantillon est de l'ordre de la centaine, sauf pour (Fer, 10^{20}), où elle est seulement de 44.

La fig. 5.6 présente la performance du test en pourcentage d'acceptation, pour $c_1 = c_2 = 1$, lorsque la seule variable à discriminer est l'énergie : pour

une gerbe attendue (Fer, 8.5E18), on compare l'échantillon de contrôle et chacune des gerbes (Fer, 1.0E19). Le test a un comportement excellent pour les gerbes Fer, mais moins bon pour les gerbes Protons où la discrimination de l'énergie n'est correcte qu'à 10%.

La performance globale peut être appréciée fig. 5.7. La combinaison des paramètres Nmuons et $N_{\max}$ sépare parfaitement les gerbes provenant de particules incidentes différentes. La dispersion du $N_{\max}$ des protons pour les deux échantillons de gerbes à 10^{19} et 10^{20} montre que la discrimination de l'énergie ne peut pas être améliorée sur ce critère.

Plus généralement, la dispersion statistique intrinsèque au processus de simulation rend probablement peu réaliste l'objectif d'un test satisfaisant à la fois en signification (faux positifs) et en puissance (faux négatifs) ; en conséquence, le test contre un échantillon de contrôle ne peut pas être la seule méthode de vérification individuelle. En revanche, accepter un nombre relativement élevé de faux positifs, permet de définir un test dont la puissance est fidèle à la la précision accessible à la simulation. Une réponse positive au test devient alors seulement une indication, qui doit être confirmée par réexécution intégrale, et cette stratégie est efficace. En effet, soit k le rapport des temps d'exécution de G et $\S(G)$; le gain du test par rapport à la réexecution est $1 - \alpha - k$; par exemple, un taux de faux positifs de 30% et un rapport des temps de 20% fournit un gain de 50%.

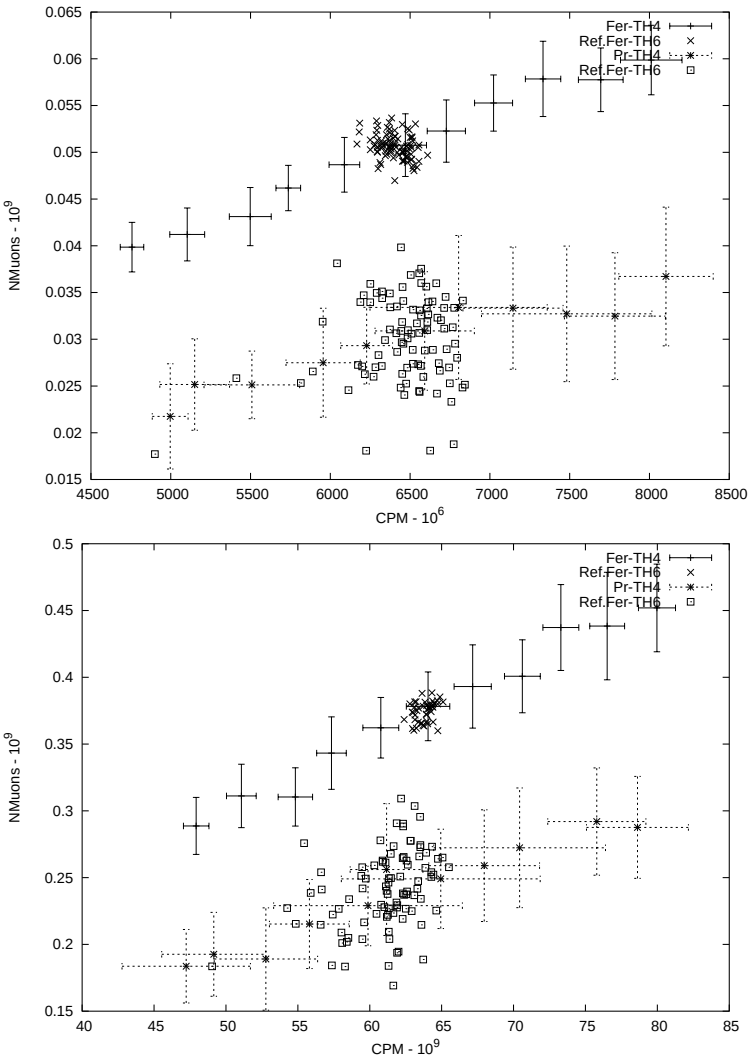


FIG. 5.7 – Distribution des gerbes. Les points avec barre d'erreur correspondent à la médiane et écart-type des gerbes de contrôle ; les points isolés représentent les gerbes éventuellement falsifiées.

Chapitre 6

Passage de messages tolérant aux fautes

L'étude du passage à l'échelle de la tolérance aux défaillances dans le modèle de calcul à passage de messages, est un project collectif des équipes Parallélisme du LRI d'une part, Clusters et Grilles d'autre part. La première étape, que j'ai initiée en 2000, a été la définition et la réalisation d'une architecture à base de mémoire de canal décrite en 6.3, qui ont été présentées dans [p30-p32]

Cette architecture a en particulier permis l'établissement d'une collaboration avec le Supercomputer Software Department de l'ICMMG (Institute of Computational Mathematics and Mathematical Geophysics) russe, matérialisée par l'accueil pendant 18 mois d'un postdoctorant dans le cadre des ACI "Accueil de jeunes chercheurs en séjour postdoctoral" du ministère de la recherche, suivi du financement d'une action bilatérale franco-russe (financement CNRS) pour 2004-2005 autour de CMDE.

6.1 Motivation et objectifs

6.1.1 Infrastructures cibles

Infrastructures de calcul global

Le modèle de programmation des environnements de calcul global est en général celui de l'exécution multi-séquentielle (embarassingly parallel, bag of tasks, etc.). Ce modèle est avant tout justifié par le rapport commu-

nication/calcul intrinsèquement défavorable d'architectures basées sur des réseaux non dédiés. Il est cependant trop limitatif : des application à couplage faible, mais non nul, de type maître-esclave ou parallèle à gros grain, peuvent raisonnablement exploiter ces architectures. L'alternative est alors entre la conception d'environnements d'exécution dédiés à des modèles spécifiques, qui sont ceux que l'on peut raisonnablement attendre sur ce type d'architecture, ou bien la réalisation d'un environnement généraliste de passage de messages supportant

- les environnements de protection des utilisateurs ;
- les défaillances inopinées et fréquentes des processus participants à l'exécution parallèle.

La première solution a été particulièrement étudiée sur des applications de branch-and-bound [112, 145]. La seconde stratégie a fait l'obejet de nombreux travaux en systèmes répartis ; [70] présente une synthèse. .

Infrastructures de grilles

L'ordre de grandeur de la fréquence des défaillances dans les infrastructures de grille est significativement inférieur à celui des systèmes de calcul global : dans le premier cas, il s'agit d'une défaillance vraie d'un processeur, d'une erreur dans l'environnement d'exécution , ou encore d'une déconnexion réseau ; dans le second cas, la récupération du processeur suffi à faire disparaître le noeud du système. Cependant, pour les infrastructures à l'échelle de plusieurs milliers ou plus de noeuds de calcul, les défaillances de processeurs ou les déconnexions réseau ne sont pas un évènement exceptionnel. Des données quantitatives dans le contextes des serveurs Web sont présentées dans [1]. D'autre part, au moins certaines de ces infrastructures sont bien adaptées à un parallélisme incluant des composants fortement couplés. Il s'agit de l'agrégation de grappes, qui constituent les composants fortement couplés, à travers un couplage à moindre performance. Le projet TeraGrid est un exemple typique de cette configuration. Il est très souhaitable de disposer d'un modèle de programmation parallèle/distribué tolérant aux défaillances pour ce type de système ; en l'état actuel des pratiques de programmation, le modèle dominant aussi bien pour les couplages forts que faibles, est le passage de message ; les modèles de programmation basés sur le paradigme du partage de mémoire étant plutôt employés pour les composants individuels des grappes lorsqu'il s'agit de multiprocesseurs.

Les configurations de protection des grilles ne sont pas unifiées, y compris

à l'intérieur d'une grille donnée : typiquement, c'est la politique de site qui définit pour chaque grappe l'autorisation sélective des connexions entrantes et sortantes.

Intermédiaires entre les systèmes de Calcul Global et les infrastructures de grille sont les systèmes de type fermes de processeurs, qui fédèrent des ressources individuelles dédiées, donc sans volatilité, à l'échelle d'un réseau local, typiquement à l'intérieur d'un domaine de protection, mais sans hiérarchie et avec un couplage faible. Enfin, les très grandes grappes fortement couplées sont également exposées au risque de défaillance d'un de leur composants.

6.1.2 MPI et tolérance aux fautes

MPI est le standard de fait pour programmer les communications des applications hautes performances, sur les grappes et les machines parallèles [108]. MPI a été conçu à l'origine exclusivement comme une librairie de communication, dans le contexte des architectures parallèles du début des années 90. Ces architectures offraient des environnements d'exécution parallèle limités, et typiquement ultra-statiques : en particulier, la soumission se fait par batch, en demandant un nombre de processeurs fixe ; la défaillance d'un processeur est donc irrécupérable, et par ailleurs improbable. Ceci a conduit à un modèle statique, MPI-1, où le nombre de processus réels qui participent à un calcul parallèle est fixe, et où les défaillances ne sont pas considérées. MPI-1 définit des contextes de communication, les communicateurs, qui incluent par exemple l'association entre identifiants de processus et leur rang (dans une numérotation de 0 à nombre de processus), ou leur index dans une topologie régulière.

La version suivante de MPI, MPI-2, fournit un support pour la création dynamique de processus, mais qui ne permet pas de gérer les défaillances. Il se limite à la création de groupes de processus MPI à communicateurs distincts, entre lesquels ne sont disponibles que des formes spécifiques de communications collectives. La défaillance d'un processus rend indisponibles les communicateurs auxquels il appartient ; la libération des ressources modélisées par le communicateur, et leur réallocation, n'est pas prévue. La communication, qui repose sur l'indexation logique, devient alors impossible.

6.1.3 Objectifs

Le spectre d'infrastructures cibles potentielles ou déjà actuelles d'un environnement de calcul parallèle tolérant aux défaillances est donc étendu, et inclut des couplages locaux fort. Ceci motive le choix de l'étude d'un environnement généraliste, par opposition aux environnements spécifiques à un modèle d'exécution de type maître-esclave.

Le choix suivant est le degré d'automaticité de la gestion des fautes. Le projet FT-MPI [87] dirigé par J. Dongarra a augmenté MPI pour d'une part détecter les fautes, aussi bien de processus que de communication, d'autre part pour permettre divers modes de réparation des communicateurs. Par exemple, l'une des options (*rebuild*) implique le démarrage de nouveaux processus pour remplacer ceux qui sont défaillants, et d'autres (*shrink,blank*) permettent de continuer l'exécution avec seulement les processus survivants. Les défaillances lors de communications collectives sont également rapportées. Cette démarche permet de minimiser la pénalité de gestion des défaillances lors des phases d'exécution normale (sans défaillance). Elle fournit au programmeur un environnement extrêmement souple, qui est typiquement bien adapté pour une application maître-esclave : la défaillance d'un esclave est détectée par le maître, qui peut répartir le travail alloué à l'esclave défaillant sur les autres.

Le modèle de FT-MPI implique que l'algorithmique de gestion des défaillances soit intégrée dans le programme applicatif. Une autre voie, qui a été choisie dans ce projet, est celle des protocoles de reprise (*rollback-recovery*). Il s'agit d'une approche *masquante* du traitement des défaillances : l'exécution, telle que vue par l'utilisateur, est strictement non affectée par les éventuelles défaillances de l'infrastructure matérielle. On verra en 6.5 qu'une approche auto-stabilisante est aussi envisageable.

Dans le contexte défini par, d'une part les architectures cibles, d'autre part le choix d'une approche transparente et masquante, les objectifs essentiels déclinent divers aspects du passage à l'échelle.

- L'adaptativité : le coût du protocole de tolérance aux fautes doit s'adapter à la fréquence et au nombre de fautes, ainsi qu'à un environnement où le comportement de fautes peut être très hétérogène. L'environnement de tolérance aux fautes peut alors être validé, au moins dans ses aspects essentiels, pour l'ensemble des infrastructures décrites ci-dessus, même si certains de ses composants ont une implémentation spécifique.
- La tolérance à un nombre de fautes simultané élevé. Dans un système de

grille interconnectant des grappes, comme dans un système de Calcul Global, un ou plusieurs processus doivent pouvoir disparaître pendant que l'environnement de tolérance aux fautes gère d'autres défaillances.
- Un modèle de ressources explicite et configurable. Les protocoles de reprise impliquent des ressources matérielles de stockage et de réseau supplémentaires pour sauvegarder de l'information relative à l'état du système. Le dimensionnement de ces ressources doit pouvoir être approximé.

6.2 Contexte : protocoles de reprise

Modèle

Le modèle classique d'un système à passage de message est un ensemble de N processus séquentiels communiquant par des canaux FIFO. La défaillance d'un processus consiste en la perte de son état et sa terminaison (modèle fail-stop). Plus précisément, les canaux correspondent à des files d'attente FIFO, avec comme opérations primitives l'émission non bloquante - dépose du message sur le canal - et la réception bloquante - attente d'un message dans la file - ; chacune de ces opérations est atomique. Le protocole de communication sous-jacent est supposé fiable : les messages ne sont ni altérés, ni dupliqués, ni réordonnées ; en particulier, une déconnexion réseau est modélisée comme une défaillance du processus qui tente de l'utiliser. L'état du système est celui de l'ensemble des processus et des canaux. L'état local d'un processus peut être sauvegardé, également de façon atomique, et utilisé pour le redémarrer. L'image sauvegardée d'un processus est un *checkpoint*.

Il faut souligner que ce modèle correspond aux processus utilisateur ; en particulier, la détection des défaillances est réalisée au niveau de l'environnement d'exécution parallèle sous-jacent (la détection des défaillances dans un modèle complètement asynchrone est d'ailleurs impossible dans le cas général).

Typologie

Les protocoles de reprise sont basés sur le redémarrage de tous, ou seulement de certains processus à partir des checkpoints, lorsqu'une erreur intervient. La difficulté est dans la reconstitution d'une état consistant, défini par

le fait que tout message qui a été reçu a été émis [46], alors que la défaillance a entraîné un état inconsistant.

Une première classe de protocoles de reprise n'est basée que sur l'état des processus, donc n'exige que des checkpoints. Les protocoles de ce type peuvent être coordonnés ou non. Dans le cas *non coordonné*, chaque processus ordonnance indépendamment ses prises de checkpoints, avec le risque de l'effet domino : il n'y a aucune garantie que le dernier checkpoint enregistré par un processus participe à un état consistant, ce qui implique la conservation de multiples checkpoints pour chaque processus. Wang et al. [194, 195] on montré que le nombre de checkpoints utiles est borné par $N(N+1)/2$, où N est le nombre de processus.

Dans les protocoles de reprise *coordonnés*, chaque ensemble de checkpoints représente un état consistant du système. Le problème est alors d'assurer qu'aucun message émis après la prise de checkpoint pour l'état consistant x par un processus i ne soit pris en compte par un processus j avant que j ait effectué sa prise de checkpoint pour l'état x. La solution la plus simple est de réaliser une synchronisation globale qui aboutit à une vidange des messages en transit avant la prise de checkpoint ; cette stratégie a été employée dans le projet CoCheck[182], et dans Condor [155, 156]. L'exemple classique de protocole asynchrone de reprise à checkpoint coordonné est l'algorithme de Chandy-Lamport [46], qui implique tous les processus, mais ne les supend pas ; les travaux ultérieurs ont tenté de minimiser le nombre de prise de checkpoints, soit en maintenant un odonnancement des checkpoints qui respecte la structure de causalité définie par les messages (modèle *Fixed Dependency After Send* [196]), soit en créant une indexation globale des checkpoints [197].

Les protocoles de reprise non coordonnés répondent bien au critère d'adaptativité : la fréquence des prise de checkpoint par chaque processus peut être définie en fonction de la fréquence des fautes attendue sur le support de ce processus. La pénalisation en espace de stockage qui est quadratique en nombre de processus les exclut.

La pénalité en fonctionnement normal des protocoles de reprise coordonnés ne provient, ni du coût de synchronisation, puisqu'il existe de nombreux algorithmes asynchrones, ni du coût de communication : les informations de contrôle (*marker* dans l'algorithme de Chandy-Lamport) sont négligeables par rapport au volume d'un checkpoint et peuvent être encapsulées dans les messages, ou sont implicites. En revanche, les protocoles coordonnés ne gèrent pas bien l'hétérogénéité des comportements de fautes, puisque la prise de checkpoint par un processus dépend du schéma de com-

munication de l'application dans son ensemble. Surtout, les protocoles de reprise basés uniquement sur les checkpoints impliquent que tous les processus soient relancés, à partir d'un état cohérent, dès qu'une faute intervient.

Les protocoles de reprise à base d'*enregistrement (logging) d'évènements* permettent de concilier l'indépendance de l'ordonnancement des prises de checkpoints et la minimisation de l'espace de stockage des checkpoints. L'exécution d'un processus est une séquence d'intervalles ne comprenant que des actions déterministes qui sont exactement reproduites si l'exécution est répétée depuis le début de l'intervalle. Les frontières des intervalles sont les évènements non déterministes, en particulier les réceptions de messages. Les émissions, étant traitées localement, ne sont pas des évènements.

En ne considérant que les communications, les protocoles de reprise à base de logging *pessimistes* sont définis par la contrainte d'enregistrement de chaque communication avant sa réception par le processus destinataire. Chaque processus peut effectuer indépendamment des prises de checkpoints et l'effet des défaillances est complètement confiné au processus fautif. La reprise s'effectue en redémarrant le processus à partir de son checkpoint le plus récent, et les réceptions sont rejouées, dans l'ordre de l'exécution avortée, pour obtenir un état strictement identique à celui qui a précédé immédiatement la faute. En particuler, la gestion des checkpoints ne demande pas de GC distribué. En revanche, l'enregistrement permanent des messages pénalise les exécutions sans faute.

D'autres protocoles existent, qui permettent d'éviter cette pénalisation. Ils augmentent le volume de checkpoints qui doivent être conservés simultanéments et implique des algorithmes de GC complexes. La réalisation d'algorithmes pessimistes semble donc au moins une étape indispensable pour évaluer la faisabilité et les performances d'une implantation de la tolérance aux défaillance pour le passage de messages dans les systèmes à grande échelle.

Une discussion détaillée du positionnement dans cette typologie des environnements de tolérance aux fautes pour MPI est présentée [27].

6.3 MPICH-V1

6.3.1 Architecture

MPICH-V1 est une réalisation d'un protocole de reprise par logging pessimiste. MPICH-V1 est formé de deux composants relativement indépendants.

Une spécialisation de la librairie MPICH. MPICH est une implémentation de MPI qui encapsule les spécificités du protocole de communication dans un composant appelé *device*. La distribution standard de MPICH offre de nombreux devices, dont *mpich_p4* pour TCP. Certains éléments du protocole de reprise sont relatifs au logging des messages et à leur traitement en cas de reprise. MPICH-V1 voit ces éléments comme un protocole de communication au même niveau, par exemple, que TCP. Un device spécifique, *mpich_cm*, a été développé. Cette démarche a permis de réaliser relativement simplement une implémentation complète de MPI 1.2.3. Le protocole de communication sous-jacent est supposé fiable (sans perte, duplication ni réordonnancement), et permet de détecter la disparition d'une connexion. L'implémentation effective est réalisée au dessus de TCP.

Un environnement d'exécution L'environnement d'exécution est responsable de la virtualisation des processus utilisateurs logiques qui composent l'application parallèle, au dessus des processus qui s'excutent sur les supports possiblement fautifs, les *tâches*. Il comporte un ensemble de services.

- Les *mémoires de canal (CM)*, comme le nom l'indique, mémorisent les messages en transit et leur séquencement. Chaque processus logique p est associé à une mémoire de canal unique (*homeCM*) qui enregistre les messages destinés à p. Tous les processus ont accès à toutes les mémoires de canal en émission.
  Les mémoires de canal ont une deuxième fonction, qui est de découpler les communications. Dans la perspective d'une communication à grande échelle, la communication directe entre tâches n'est pas toujours réalisable : elle implique, soit une perte de performances sévère si les messages sont transportés à travers un protocole généralement accepté (http ou soap), soit qu'au moins un des deux partenaires soit prêt à accepter des connexions, typiquement TCP, ce qui est souvent rendu impossible par la configuration des pare-feux. Les mémoire de

canal doivent donc pouvoir fonctionner comme un relai (proxy) de communication.
- Le *système de sauvegarde* déclenche les prises de checkpoints sur chaque tâche, et enregistre de façon permanente l'image mémoire des tâches. L'ordonnancement des déclenchement est indépendante sur chaque tâche. L'enregistrement est non-bloquant, par un mécanisme de clonage : chaque tâche peut continuer à calculer pendant qu'un clone voit son image enregistrée et stockée.
- Le *Dispatcher* coordonne l'utilisation des ressources. Il est responsable de la supervision des tâches, en particulier de la détection des défaillances et de la réalisation du modèle fail-stop ; de la création dynamique de tâches, de leur association avec les processus logiques ; de la mise en relation des services mémoires de canal et serveurs de checkpoints. Les services sont décrits par un Service Registry, qui définit les hôtes qui fournissent ce service et les informations nécessaires pour effectuer cette mise en relation.

6.3.2 Les mémoires de canal

Les mémoires de canal remplissent une double fonction : stocker les messages en transit et découpler les communications. Pour remplir le deuxième rôle, elles doivent être supportées par des machines qui acceptent les connexions en provenance des noeuds d'exécution. Toutes les connexions entre les noeuds de calcul et les mémoires de canal sont initiées par les noeuds.

Au lancement, une tâche établit un canal TCP avec toutes les mémoires de canal du système ; le nombre de connexions pourrait d'ailleurs être diminué en exploitant la stucture des communicateurs MPI. Pour émettre, une tâche dépose un message qui contient les données du message MPI dans la mémoire de canal homeCM du destinataire ; pour recevoir, elle émet une requête et reçoit les informations correspondantes en retour.

Pour chaque émetteur p_j, la mémoire de canal HomeCM de p_i maintient une FIFO qui réalise la mémorisation de la séquence des émissions, donc des réceptions, puisque MPI impose que les canaux entre deux processus soient FIFO. Une FIFO commune à tous les émetteurs mémorise la séquence des messages de contrôle, en particulier ceux qui correpondent à une requête sur un émetteur non nommé.

Le checkpoint d'un processus contient le numéro logique du dernier message émis en direction de chaque processus. A l'initialisation d'une reprise, la

nouvelle tâche peut recevoir le nombre de messages émis par l'exécution interrompue du processus depuis le dernier checkpoint, lorsqu'elle se reconnecte aux mémoires de canal ; ces messages ne doivent pas être émis à nouveau ; une conséquence importante est que la réexecution est plus rapide que l'exécution initiale. L'initialisation pour la réception des messages en reprise consiste à déplacer le pointeur de messages à délivrer en tête de la FIFO. Lorsqu'une prise de checkpoint est complétée pour un processus, les messages reçus par ce processus peuvent être éliminés.

6.3.3 La librairie MPICH-CM

Les mémoires de canal communiquent directement avec les tâches, à travers la librairie spécialisée ; elles réalisent en particulier la fonction des tampons de réception qui stockent habituellement sur les noeuds récepteurs les messages en attente de requête de réception par la tâche ; MPI propose en effet un ensemble de modes de communications impliquant en particulier des réceptions non-bloquantes. Un device MPI 1 se réduit à quatre fonctions : émission, réception, et deux fonctions d'inspection (*from* et *probe*). L'implémentation de ces fonctions de communication suit le même schéma : un message de contrôle notifie à la CM le type de communication et les propriétés du message (*source*, *destination*, *size*, *tag* and *kind*). Ensuite, le contenu du message est transmis.

6.4 Performances

6.4.1 Surcoût de communication

Cette section décrit le surcoût du système des mémoires de canal en fonctionnement sans faute, sur les fonctions élémentaires de communication. L'environnement d'expérience est la grappe de PC du LRI (500 to 700MHz en 2001) connectés par un switch 100 Mbit/s et par noeud.

Le temps d'une communication point-à-point aller-retour (RTT) bloquante entre deux noeuds communiquant à travers un serveur de mémoires de canal est décrit fig. 6.1, et comparé à l'implémentation de référence sur TCP (device *ch_p4*). La latence est presque exactement double de celle de l'implémentation de référence : les deux communications requises par le passge de la mémoire de canal sont séquentialisées, et le passage par la mémoire de canal est

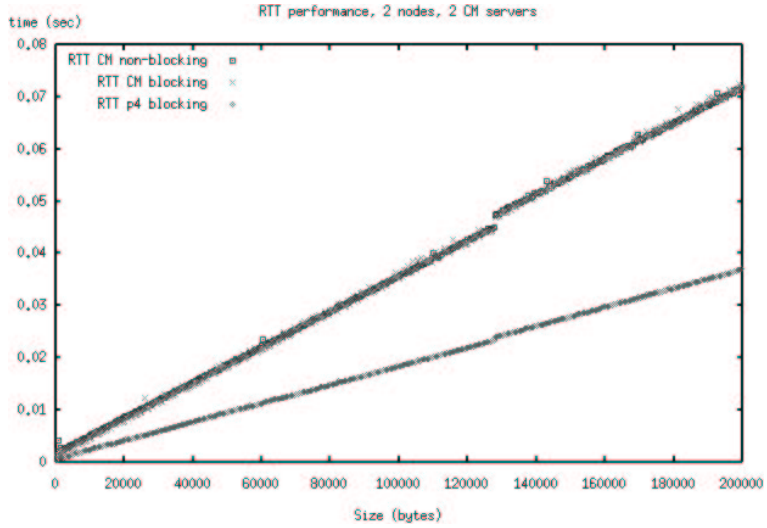


FIG. 6.1 – Comparaison des performances d'un ping-pong pour les devices ch_cm et ch_p4.

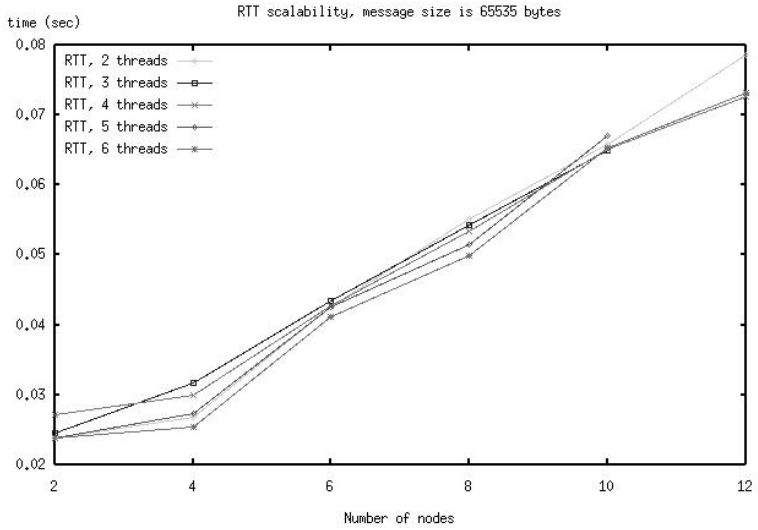


FIG. 6.2 – Influence de la charge des mémoires de canal. Les courbes correspondent au niveau de multi-threading de l'implémentation ds mémoires de canal.

négligeable. Une partie de cette pénalité est évitable en pipelinant la transmission des messages longs lorsqu'une réception est postée. Cependant, le comportement linéaire montre un débit quasi-constant, de l'ordre de 20Mb/s pour MPICH-V et 40Mb/s pour la référence, qui correspond probablement à un goulet d'étranglement au niveau du protocole sous-jacent. Dans ce cas, le pipeline n'améliorera pas la performance, qui est limitée par le débit de communication de l'infrastructure du serveur de mémoire de canal.

Cette hypothèse est vérifiée par l'étude de l'extensibilité presentér fig. 6.2. Un seul serveur supporte les mémoires de canal qui desservent des paires de processus qui communiquent indépendamment. Le comportement quasi linéaire à partir de 4 processeurs montre que la limitation en débit est dominante. L'intérêt d'une architecture multithread pour la mémoire de canal est modéré. On retrouve un comportement analogue dans les comunications collectives. Une analyse plus détaillée est présentée dans [171].

6.4.2 Un modèle global

On veut définir un modèle de performance, qui prenne en compte à la fois la pénalité de découplage des communications, le coût des redémarrages, et le gain obtenu par le protocole de reprise.

Sauvegardes périodiques

Comme dans le chapitre précédent, les instants des fautes forment un processus ponctuel $t_0 = 0, t_1, t_2, \ldots, t_n, \ldots$. Dans la suite, on considère un processus de Poisson de paramètre λ.

La figure 6.3 décrit la modélisation du processus de reprise sur un process (au sens de processus abstrait dans les sections précédentes, pour éviter l'ambiguité avec le processus ponctuel des fautes). Les prises de checkpoints sont périodiques, de period T ; le coût d'une prise de checkpoint est o_c. Si une faute intervient pendant une période, le process redémarre à partir du dernier checkpoint, après un délai de rechargement o_r ; tout le travail effectué dans cette période jusqu'à la faute est perdu. o_c and o_r sont petits par rapport au délai inter-arrivée $1/\lambda$. Les prises de checkpoints et redémarrages sont supposées sans faute.

Le coût de référence est celui qui correspond à une exécution sans faute, noté C. Sur l'intervalle $((k-1)T, kT]$ (période k), soit τ_k l'instant de la

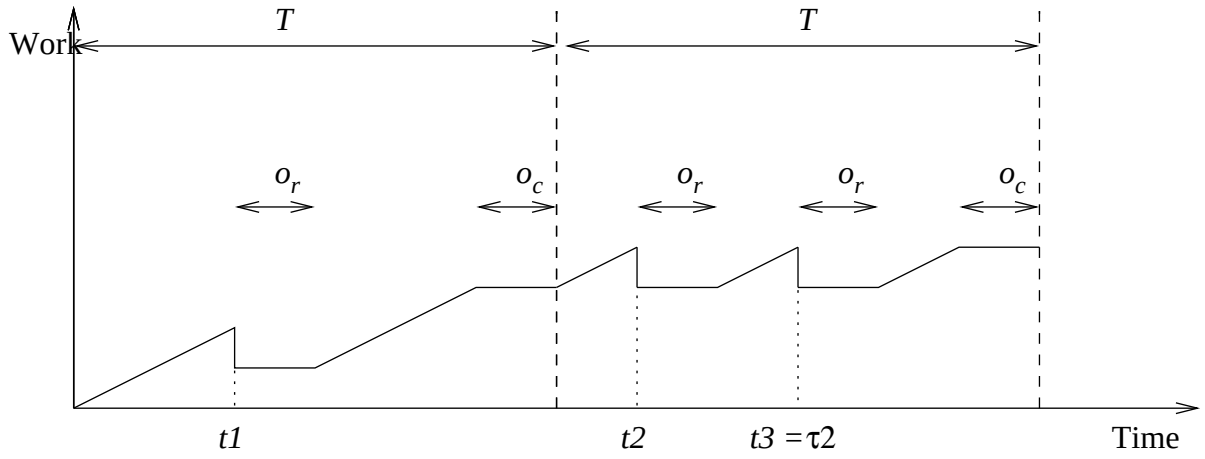


FIG. 6.3 – Modélisation du coût des sauvegardes périodiques

dernière faute, ou 0 s'il n'y a pas de faute. Tous les τ_k on la même loi, à une constante additive près.

Soit w_k le travail utile effectué dans la période k. On a

$$w_k = T - o_c \text{ si } \tau_k = 0$$

$$w_k = T - \tau_k - o_c - o_r \text{ si } \tau_k > 0$$

Le travail total effectué jusquà la période n incluse est noté $W_n = w_1 + w_2 + \ldots + w_n$. Les w_k sont iid ; soit w une variable de même loi. En utilisant des techniques analogues à celles présentées dans la première partie, on obtient une approximation du ralentissement moyen, l'approximation étant valable pour C grand.

Plus précisément,

$$E(w) = \alpha - \beta e^{-\lambda T}$$

avec $\alpha = \frac{1}{\lambda} - o_c - o_r$ et $\beta = \frac{1}{\lambda} - o_r$.

Soit $M(x)$ le processus de comptage associé au processus défini par les w_k, *i.e.* $M(x)$ est le nombre de w_i inférieurs à x. Les w_k forment un processus de renouvellement, donc $\lim_{x\to\infty} \frac{E(M(x))}{x} = \frac{1}{E(w)}$. Pour C grand, on obtient $E(M(C))E(w) \approx C$. L'approximation est utilisable pour des valeurs réalistes du temps de référence, typiquement moins de 10λ, alors que λ représente la fréquence d'arrivée des fautes sur l'ensemble du système.

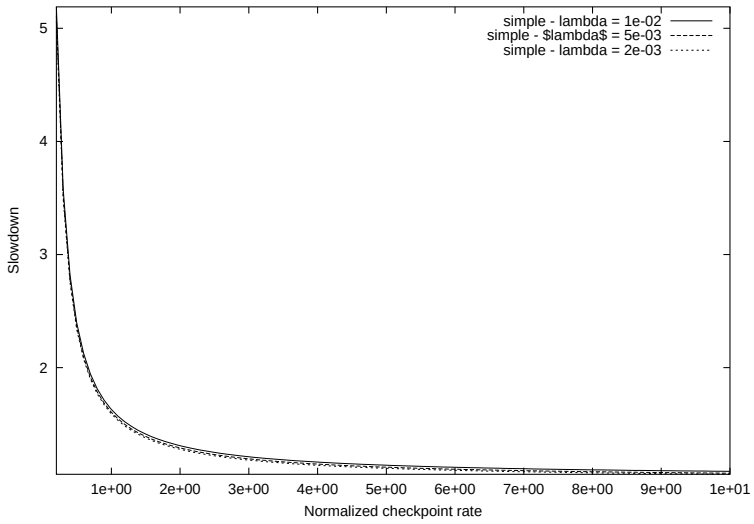


FIG. 6.4 – Surcoût dû au rechargement.

Le temps d'exécution moyen pour une période de prise de checkpoint T est $M(C)T$; le rapport entre ce temps d'exécution et le temps de référence est le ralentissement $r(T, \lambda)$:

$$r(T, \lambda) = \frac{T}{\alpha - \beta e^{-\lambda T}}$$

Il est facile de vérifer que r a un minimum, qui correspond à la période optimale. Quand T tend vers l'infini, r est équivalent à T/α, donc linéaire en T.

Surcoût de sauvegarde

L'étude expérimentale [29], indépendante de ce qui est présenté ici, fournit des valeurs numériques raisonnables pour évaluer quantitativement l'impact des prises de checkpoint. [29] indique que, dans le contexte favorable d'un réseau local et de machines dédiées, la pénalité d'un rechargement o_r varie linéairement entre 3 secondes pour un accès isolé à 24 secondes pour 25 accès concurrents lorsqu'un seul serveur de checkpoint est présent.

Pour MPICH-V1, on peut considérer $o_c = 0$, la prise de checkpoint étant clonée. Dans ce cas, le ralentissement est une fonction croissante de T (l'optimum est pour $T = 0$). Pour ces paramètres, la figure 6.4 montre le ralentis-

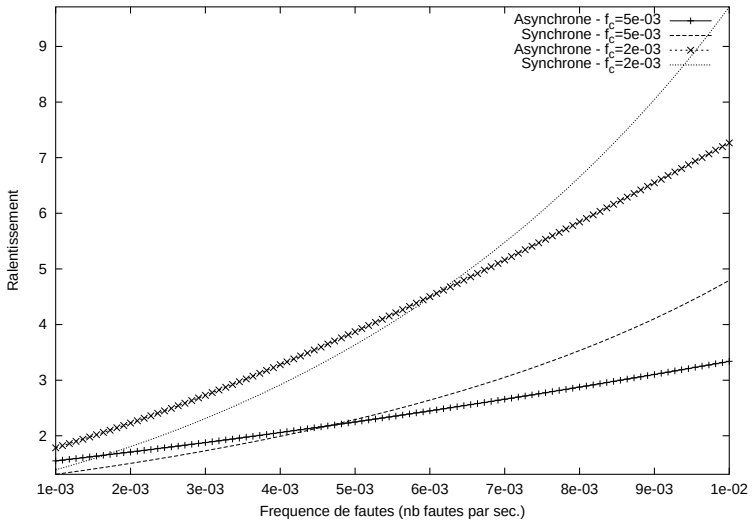


FIG. 6.5 – Comparaison des méthodes de reprise

sement en fonction du nombre de prises de checkpoints dans le délai moyen entre deux fautes (fréquence normalisée λ/T), dans l'hypothèse d'un délai de 3 secondes. Pour les valeurs considérées de la fréquence des fautes λ, les résultats sont très proches : typiquement, pour une fréquence de checkpoint égale à celle des fautes, le temps d'exécution est un peu moins que doublé, alors qu'avec une fréquence double, le pénalité est de l'ordre de 30%.

Comparaison de protocoles

Le modèle précédent permet de comparer les performances d'un protocole coordonné à la Lamport, et d'un protocole à logging de messages. Pour évaluer une stratégie à logging de messages, l'impact du logging des messages sur l'application dépend du rapport communication-calcul de l'application, en faisant l'hypothèse simplificatrice que les communications et les prises de checkpoints ne sont pas en conflit pour l'accès réseau. Cet impact peut donc être approximativement modélisé ou expérimenté pour chaque application particulière, en fonction des paramètres de base de l'application et du protocole, suivant les modèle classiques de performances en parallélisme [77].

Dans un protocole synchrone, à la Lamport, il n'y a pas de pénalité de logging, mais la pénalité de prise de checkpoint ne peut plus être nulle : l'ensemble des process doivent se synchroniser pour la sauvegarde. Un modèle

très simple des protocole synchrone consiste à considérer $o_c = o_r$. D'autre part, la pénalité de prise de checkpoint est plus importante, à cause de la congestion créee par la probable simultanéité des accès au service de sauvegarde. La figure 6.5 propose une comparaison des stratégie stratégies, pour les paramètres suivants : ralentissement dû aux communications 40%, et coût de sauvegarde dans le cas synchrone 24 secondes. Le surcoût de commmunication pénalise la stratégie à logging lorsque les fautes sont rares, mais le coût moindre de synchronisation intervient lorsque les reprises sont fréquentes. Pour des valeurs plus faibles de la pénalité de communication (de l'ordre de 10%), la stratégie à logging est toujours meilleure.

6.5 Conclusion

Ce qui précède a montré qu'il est possible d'intégrer un protocole de reprise à base de logging de messages dans MPI. La réalisation à base de mémoires de canal fournit en outre un mécanisme de découplage des communications qui permet la cohabitation avec des politiques de protection strictes. Plusieurs études resteraient à développer pour réaliser les objectifs présentés au début, qui sont l'adaptativité, la tolérance à un nombre de fautes simultané élevé, et un modèle de ressources explicite et configurable.

6.5.1 Des services tolérants aux défaillances

L'environnement comprend des services dont la défaillance est fatale dans le schéma présenté ci-dessus. Les services de dispatch et sauvegarde sont réalisables simplement en un modèle transactionnel. En revanche, le service de mémoire de canal ne peut pas l'être ; le protocole assure certes l'atomicité vue par les process des échanges avec la mémoire de canal ; mais, sauf à imposer une pénalité insupportable aux communications, l'enregistrement des messages doit s'effectuer en mémoire et non sur disque. Tout dépend alors de la configuration cible. L'exécution de programmes parallèles relativement courts sur des environnement très volatils peut faire l'hypothèse de serveurs de mémoires de canal stables : il s'agit alors de machines dédiées ; elles doivent être en petit nombre, ce qui implique que les communications qu'elles sauvegardent ne sont pas trop fréquentes, à la fois pour des raisons de performances - le ralentissemnt croît linéairement avec le nombre de mémoires de canal par serveur - et d'encombrement.

On verra en 6.6 comment diminuer significativement la charge en messages. Une autre solution, basée sur un protocole de sauvegarde uniquement des informations de causalité, est présenté en [28], mais il ne permet pas le découplage des communications. La tolérance aux défaillances des serveurs de mémoires de canal sans augmenter la pénalité des communications implique probablement des sauvegardes coordonnées comme dernier recours. Une analogie peut être trouvée dans les algorithmes complètement distribués pour l'exécution de lot de tâches indépendantes (problème du DO-ALL) [69, 49]. [69] gère uniquement la disparition des process, en diffusant l'information sur les tâches effectivement exécutées à des sous-ensembles plus souvent qu'à la totalité des process ; [49] étend cet algorithme à l'intégration de nouveaux process. Pour la communication, les mémoires de canal sont l'analogue de la diffusion sur des sous-ensembles, la sauvegarde coordonnée à la diffusion totale ; le rapport optimal entre la fréquence des deux types de sauvegarde pourrait être évalué à partir du modèle présenté en 6.4 ; il mesurerait un équivalent de la *laziness* de [197].

6.5.2 Ordonnancement des sauvegardes

Le problème de l'ordonnancement des sauvegardes pour les programmes séquentiels a été modélisé dans [51] et étendu par Rosenberg [165, 163]. Ces travaux font l'hypothèse d'une distribution statistique connue des fautes sur un, ou un ensemble de processeurs. En particulier, Rosenberg a obtenu des résultats d'optimalité pour plusieurs distributions classiques. Bien que la modélisation présentée ci-dessus soit beaucoup plus élémentaire, elle repose sur les mêmes hypothèses essentielles, celles d'un côut constant des opérations de gestion des process, sauvegardes et relances, et de l'indépendance entre sauvegarde et communication.

Cette hypothèse n'est pas complètement réaliste dans le cas d'un système à grande échelle, surtout pour une stratégie avec logging de messages. Les communications et les sauvegardes s'adressent toutes les deux in fine à la ressource réseau ; la sauvegarde en progrès d'un processus va donc ralentir les communications des process qui partagent la même ressource. D'autre part, la réexécution n'est pas identique à l'exécution initiale, puisque certaines communications sont évitées.

Pour prendre en compte les conflits de ressource, on peut envisager de modéliser les communications d'une part, les prises de checkpoints d'autre part, en fonction de leur degré de simultanéité. Les surcoûts pourraient alors

être modélisés par des variables aléatoires, dont la loi dépend du processus de fautes. Il est cependant peu probable que cette approche puisse déboucher sur une modélisation analytique complète : il est en effet connu que la superposition de requêtes simultanées sur une ressource peuvent aboutir à un temps de réponse de type heavy-tailed. [132, 149].

Plus généralement, les travaux de Rosenberg ont montré que, pour un problème relativement simple, à coûts constants, les ordonnancements optimaux sont très difficiles à obtenir. D'autre part, on a vu dans la section précédente que, pour un ordonnancement périodique, et les mêmes hypothèses de coûts constants, les performances sont extrêmement sensibles à une sous-estimation de la période de sauvegarde. Pour exploiter réellement la capacité des stratégies à logging à gérer l'hétérogéinéité des comportements de fautes, il faut intégrer une stratégie d'ordonancement dynamique des sauvegardes au niveau des processus. Le profilage d'une exécution reste à définir : il doit clairement prendre en compte, d'une part la fréquence des fautes, d'autre part, le travail utile effectué par l'application. Mesurer cette dernière quantité peut s'envisager sous la forme d'indications au niveau du programme.

6.6 Perspectives : itérations asynchrones et tolérances aux fautes

L'approche masquante considérée jusqu'ici est nécessaire lorsque l'application n'exhibe pas de propriété specifique. A l'inverse, le coût en ressources matérielles et en performances impliqué par la simulation d'une exécution sans fautes n'est pas impératif lorsque l'application est capable de récupérer spontanément un comportement correct à partir d'une perturbation. Le contextes correspondants sont l'auto-stabilisation et les modèles d'itérations asynchrones.

Les modèles d'itérations asynchrones, initiés Chazan et Miranker sous le nom de relaxation chaotiques [48], étendus par Bertsekas et al. [20, 168] et Uresin et al. [190] portent sur les méthodes itératives. [67] a montré leurs relations avec le formalisme de l'autostabilisation.

Une méthode itérative synchrone est une récurrence $x^{(k+1)} = f(x^(k))$

$$\begin{array}{lll} f : X_1 \times \ldots X_n & \rightarrow & X_1 \times \ldots X_n \\ x = (x_1, \ldots, x_n) & \rightarrow & f_1(x), \ldots f_n(x) \end{array}$$

et la fonction f est telle que l'itération converge à partir d'une valeur initiale quelconque (ou dans un sous-ensemble). Le parallélisme intervient au niveau du calcul des x_i, qui sont indépendants. Si le process i gère la composante x_i, les communications correspondent à l'acquisition à l'itération k des composantes x_j pour $j \neq i$. Pour qu'une parallélisation à passage de messages soit réaliste, x_i ne doit bien sûr dépendre que d'un sous-ensemble des autres composantes.

Les modèles d'itérations asynchrones relaxent la contrainte de synchronisme en montrant que l'itération peut converger même si la mise à jour de chacune des composantes se fait avec des valeurs périmées des autres composantes : $x_i^{(k+1)} = f_i(x_1(\tau_1^i(k)), \ldots, x_n(\tau_n^i(k)))$, où les instants k ne représentent plus un indice global d'itération, mais l'union de touts les instants des itérations locales.

Le modèle d'itérations asynchrone permet de représenter le fonctionnement sans défaillances d'un système à passage de messages à réception non-bloquantes : les valeurs non locales utilisées pour l'itération locale sont celles de la dernière réception effective. Lorsqu'il y a réception, $k - \tau_j^i(k)$ représente le délai du message envoyé par le process j vers le process i. Une des conditions de convergence est que le système finisse par purger toute information périmée, donc que la séquence des arrivées de messages sur chaque process ne soit pas finie. En échange, les conditions sur la fonction d'itération sont plus fortes que pour les itérations synchrones, analogue à l'axiome d'indépendance en algorithmique distribuée ; pour les applications numériques, f doit être contractante pour la norme du sup.

Le modèle du calcul asynchrone permet aussi de représenter la reprise avec pertes de messages. A notre connaissance, cette application n'a pas été envisagée précédemment. Chaque process effectue indépendamment des prises de checkpoints. Le reprise d'un process s'effectue uniquement à partir du checkpoint, et correspond à une itération à partir des valeurs que possédait le process immédiatement avant la prise de checkpoint. Le logging des messages disparaît complètement.

Ce schéma de reprise viole la causalité au sens de Lamport, puisque les messages envoyés par ce process pendant le dernier intervalle inachevé de checkpoint sont orphelins. Dans le cas d'un schéma itératif, les messages orphelins diminuent fortement la pénalité sur la convergence induite par la perte des calculs ; ils constituent une trace de l'avancement de l'itération sur le process défaillant, qui est indirectement mémorisée par son influence sur les itérations des autres process.

L'importance de ces corrélations indirectes est évidemment complètement dépendante de l'application. Elles pourraient conduire à des contraintes encore plus faibles, en limitant aussi la nécessité de prise de checkpoint. Un exemple jouet le montre. On considère une itération de type Red-Black Gauss-Seidel pour la solution de l'équation de Laplace avec second membre nul, dont la version synchrone est :

$$x_{ij}^{(k+1)} = \frac{1}{4}(x_{i-1,j}^{(k)} + x_{i+1,j}^{(k)} + x_{i,j-1}^{(k)} + x_{i,j+1}^{(k)} \; 0 < i, j \leq N,$$

les bords sont fixés à 0 ; l'état initial du la matrice est aléatoire.

La matrice des x_{ij} est partitionée par blocs ; le schéma de faute a la structure représentée fig. 6.6 portant sur les blocs ; les fautes sont simultanées et la reprise s'effectue à partir des valeurs initiales. La vitesse de convergence dépend bien sûr de la fréquence spatiale des fautes, mais aussi du rapport périmètre/surface de blocs : un bloc est d'autant plus "perméable" à la rectifications par ses voisins qu'il est petit.

La plupart des travaux sur le placement-ordonnancement à grande échelle (par exemple [58, 16]) admettent implicitement des communications à réception bloquantes ; l'écart de performance de communications entre les réseaux locaux et les réseaux à longue distance tendrait alors à limiter l'extensibilité à des applications de type couplage, où les noyaux de calcul sont exécutés sur des grappes fortement couplées. Le modèles des itérations asynchrones permet d'envisager d'étendre le champ des applications du calcul à grande échelle et d'unifier en partie les problématiques de placement-ordonnancement d'une part, de tolérance aux défaillances d'autre part : les écarts de débit de communication n'interdisent pas complètement le progrès de l'application, et une faute apparaît comme un ralentissement supplémentaire, jusqu'à ce que la reprise rétablisse le flot de messages. Des algorithmes d'analyse numérique [34] ou de simulation en dynamique moléculaire (méthode de Verlet) utilisent d'ailleurs des approximations qui correspondent à un asynchronisme contrôlé au niveau utilisateur.

Avancer dans ce domaine suppose de définir une métrique pour les applications, ou par classes d'applications, qui quantifie la capacité de l'application à progresser malgré les perturbations. Le concepteur ou l'utilisateur de l'algorithme peut exploiter cette métrique pour effectuer des choix infomés sur le type d'architecture cible ou les ressources nécessaires. Quelques résultats existent, pour le cas général [43], et pour des applications particulières [34], mais ils ne prennent en compte que que les délais de communi-

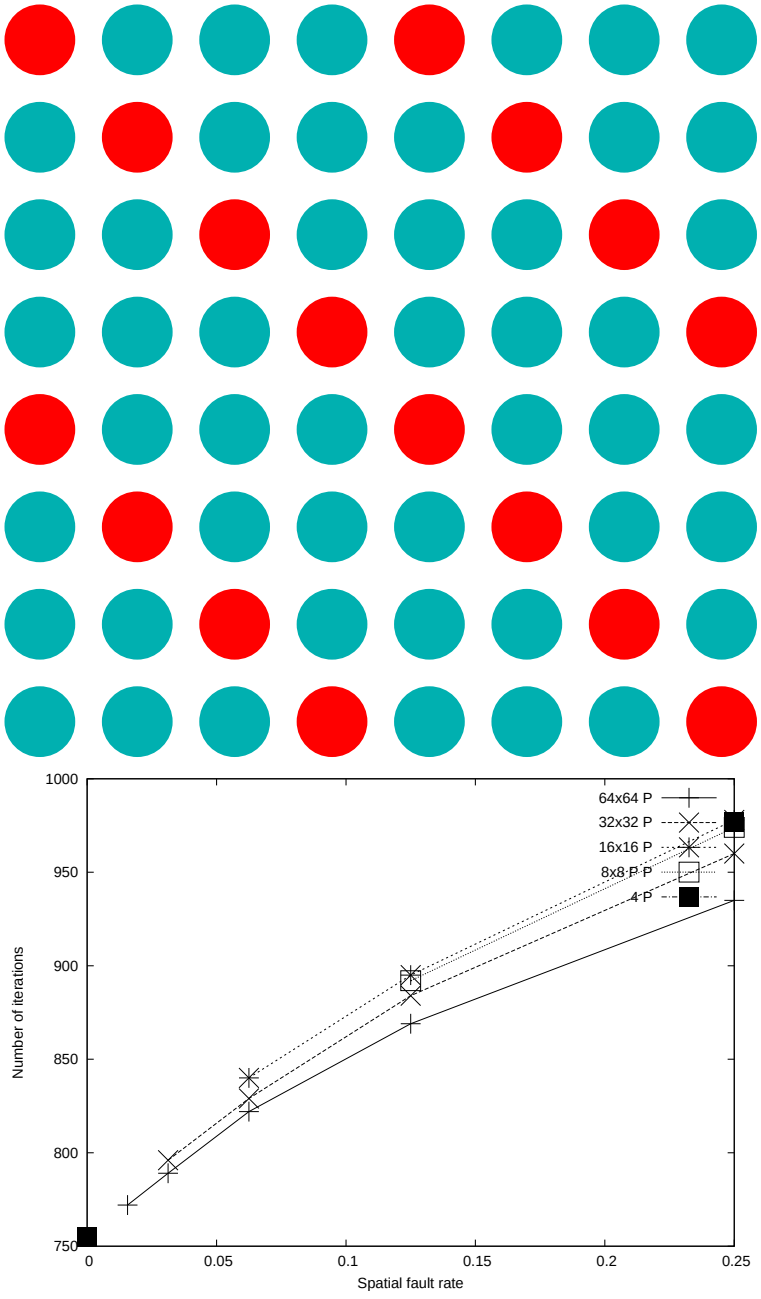


FIG. 6.6 – Exécution du schéma de Jacobi avec fautes, matrice 1024×1024.
1 faute toutes les 500 itérations.

cation et la vitesse de convergence intrinsèque de de l'algorithme et non sa capacité spécifique d'autostabilisation.

Des méthodes de point fixe, donc des itérations asynchrones, sont très fréquentes au niveau le plus interne d'applications numériques et non-numériques parallèles, par exemple les algorithmes de résolution itérative de systèmes linéaires, ou la recherche de plus courts chemins, et constituent l'essentiel du coût de calcul et du volume de communication. Dans une application réelle, ils sont toujours intégrés dans des schémas de calcul qui ne sont pas ce type. Une approche mixte devrait limiter le protocole de reprise à logging de message à ces segments du calcul.

Chapitre 7

Perspectives : interactivité et grilles

7.1 Introduction

Les systèmes de grilles ciblent la catégorie de services qui requièrent des ressources importantes en calcul, en stockage et en réseau. Un des objectifs essentiels est de traiter des quantités considérables de données, produites dans des contextes variés. Pour certains de ces contextes, par exemple la physique des particules, ou l'analyse d'images radiologiques 2D dans le cadre d'études épidémiologiques, les algorithmes de traitement des données impliquent peu d'interaction homme-machine, encore que cette affirmation doive être nuancée (7.2.1). L'objectif majeur est de fournir un accès rapide et élaboré à du stockage distribué à l'échelle, et l'analyse automatique de données, ou bien la réalisation de simulations produisant des données massives. Dans ce cas, les grilles fonctionnent comme un très grand centre de calcul, avec un modèle d'exploitation en soumission par lot (*batch*).

Pour d'autres applications, la compétence du système visuel humain à reconnaître et interpréter des structures complexes reste utile ou même incontournable. Le post-traitement au sens large, incluant une composante de visualisation, peut alors s'inscrire dans une boucle avec le système de traitement supporté par la grille, ce qui impose à ce système une contrainte d'interactivité. Pour la satisfaire, les services de la grille doivent servir les requêtes dans des limites de temps compatibles avec un dialogue homme-machine adapté à l'application considérée. La grille est alors vue comme une

ressource de grande puissance à laquelle l'utilisateur accède de façon totalement transparente à partir de son environnement habituel, celui du poste de travail individuel. Ce modèle d'exploitation est encore très peu développé. Typiquement, dans le domaine de la production sur grilles, les applications de la physique des hautes énergies sont en phase de préproduction, alors qu'il n'existe à notre connaissance aucune application interactive qui ait atteint ce niveau.

La transparence de l'accès à la puissance de calcul et aux données est la métaphore fondatrice des grilles. L'intégration sans rupture de la puissance de la grille dans l'utilisation banalisée du poste de travail, lorsque sa capacité devient insuffisante, demande de proposer à tous les niveaux de l'intergiciel de grille des protocoles nouveaux, orientés non plus vers le débit, mais vers le temps de réponse, en conservant les acquis des technologies de grille qui permettent de construire des organisations virtuelles : globalisation des ressources, sécurité, login unique etc. Ces protocoles doivent offrir une QoS pour les applications interactives, tout en garantissant une exploitation efficace et équitable des ressources pour toutes les applications, interactives ou batch.

Ce chapitre présente plutôt une analyse de la problématique et des perspectives de recherche que des résultats publiés, sauf dans la section 7.4, qui concerne l'imagerie médicale. Il est associée aux activités suivantes.

- L'ACI Masse de Données AGIR *Analyse Globalisée des Images Radiologiques* dont je suis responsable.
- Ma participaton au projet EGEE.
- Les publications [p10, p33,p35, p37-p39]

7.2 Interactivité et grilles

7.2.1 Deux exemples d'applications

Imagerie médicale

Les systèmes d'imagerie médicale (Scanner, IRM, PET-Scan) sont parmi les plus gros créateurs de données numériques. L'évolution des dispositifs d'imagerie médicale, par exemple les scanners multi-détecteurs et l'IRM fonctionnelle, ont conduit à une explosion de données sur les patients : un scanner spiral multi-détecteur produit typiquement une image de 1000 coupes, chacune de $512 \times 512 \times 16$ bits, et un examen comprend plusieurs images.

Les nombreux travaux en traitement d'images médicales des quinze dernières

années ont développé des méthodes, des algorithmes et des environnements largement validés au niveau médical ; cependant, le fossé entre ces travaux et leur adoption en pratique médicale (clinique, recherche, épidémiologie) reste massif.

Une des explications possibles de cet écart réside dans la contradiction entre deux modèles de production : celui de l'informatique médicale est marqué par l'interactivité, l'intégration dans des chaînes de traitement spécifiques (PACS), et la standardisation au niveau d'ordinateurs personnels ; alors que les applications de traitement d'images requièrent souvent des environnements haute performance et/ou l'accès à des données distribuées, ce qui relève du modèle de production centre de calcul ou de technologies avancées. Plus fondamentalement, en l'état de l'art actuel, certains algorithmes ou cas d'application requièrent *à la fois* des capacités de calcul (au sens large) de hautes performances et de la supervision interactive [114]. Par exemple, la supervision interactive du calcul permet d'éviter les minima locaux dans certains problèmes difficiles en recalage non linéaire ou en segmentation. L'intégration de l'interactivité et des grilles serait une opportunité pour transférer les travaux de recherche en imagerie médicale vers la pratique clinique, en fournissant l'accès transparent qui correspond au mode de travail des médecins à des ressources de calcul et d'information étendues.

Physique des particules

La communauté de la physique des particules développe actuellement une infrastructure matérielle et logicielle de grille, à travers les projets européens EDG, LCG et EGEE, destinée à l'analyse des données qui seront produites par le futur accélérateur LHC (Large Hadron Collider) à partir de 2007. La quantité énorme de données attendues, de l'ordre des PBytes par an, et le caractère mondial de la collaboration, ont conduit au choix d'une approche très distribuée.

Cette communauté, traditionnellement utilisatrice d'une informatique très centralisée, commence a ressentir la nécessité d'une meilleure intégration entre outils locaux et ressources de hautes performance (fig. 7.1). Il est d'ailleurs caractéristique que l'environnement AliEn, qui a été développé pour les besoins de l'expérience de physique ALICE, soit également à la base du projet européeen d'analyse d'images radiologiques MammoGrid.

While the existing processing schemes using batch queues and mass storage systems have proven in the past that large datasets can be handled very well inside individual computer centres,scaling this model to the global scale presents new and formidable computing challenges. AliEn can be used today to solve typical HEP use cases like running simulation, reconstruction and subsequent centrally coordinated processing steps as it was outlined above. It is likely that a large fraction of the CPU time use in HEP will actually be spent in executing these steps. *However, end users will be waiting for output of these organized production steps to start a largely chaotic process of analysis where many of them (hundreds) will be trying to run their (possibly private) algorithms over a large subset of distributed datasets with the aim to extract the physics observables specific to their analysis.*

FIG. 7.1 – Citation de [32]. Les passages soulignés le sont par nous.

7.2.2 Scénarios

Les étapes d'une interaction à base de visualisation ont été décrites dans [114]. Pour les applications décrites ci-dessus, elle se spécialisent de la façon suivante.

Dans le cas de l'imagerie médicale, l'*acquisition* des données est réalisée par un dispositif installé en milieu hospitalier, et généralement encodée suivant le standard DICOM. Pour la physique des particules, les données peuvent provenir d'un dispositif expérimental ou de simulations numériques.

L'*exploration* correspond à la navigation dans les données et les métadonnées associées ; pour certaines applications d'imagerie médicale, c'est un processus largement interactif faisant intervenir la compétence du praticien (choix des images ou des régions importantes) ; pour d'autres applications, le processus est automatique.

L'*analyse* traduit les données brutes vers les quantités nécessaires pour répondre aux questions de l'utilisateur. Les tâches peuvent être simples, comme les projections planaires utilisées pour la navigation dans une image, ou très coûteuses en temps, pour certains algorithmes de segmentation ou en fusion multimodale.

L'*interprétation* génère une représentation visualisable des données, par exemple la recontruction d'un volume, et peut être coûteuse en temps de calcul.

La localisation des étapes finales, rendu et rasterisation, dépend de l'ap-

plication et de l'environnement cible. Pour des images très lourdes, ou si la cible d'interaction est un environnement de réalité virtuelle et/ou de travail collaboratif, cette phase peut demander une architecture distribuée ; dans d'autres cas, un poste de travail convenable suffit.

Pour la physique des particules, les composantes essentielles seront la navigation et l'analyse, les interfaces d'interprétation étant en général assez simples, du type histogrammes.

Les étapes d'acquisition et de rendu sont relativement disjointes des autres ; quelques exemples de travaux dans le domaine sont [118, 174]. L'interaction avec l'utilisateur est présente dans les étapes d'accès, analyse, et interprétation. Elle est considérée à la fois du point de vue de la navigation dans les données, qui sont partagées entre la ressource locale et celles distantes, et du point de vue du contrôle du traitement (calcul), qui est initié, et éventuellement modifié, en temps réel.

Le scenario le plus simple pour coupler interactivité et grilles est celui où les données sont localisées à proximité de l'utilisateur, sur sa machine ou sur un réseau local ; le pré- et le post-traitement, ainsi qu'une partie de l'interaction, sont locaux ; seules les parties les plus consommatrices de puissance de calcul dans l'analyse et l'interprétation, sont déportées. Ce schéma sera applé dans la suite *Local Data Remote Computation* (LDRC).

Un scenario plus complexe, *Remote Data Remote Computation* (RDRC), est celui où les données sont stockées à distance dans une base de données centralisée ou distribuée, et peuvent également être répliquées [2, 125, 76, 109, 33] ou dans des caches (DPSS/IBP [15]). La navigation, l'analyse et l'intreprétation sont complètement déportées et l'utilisateur interagit à travers une vue locale partielle. Les obstacles à la réalisation de l'interaction dans ce scenario proviennent à la fois du volume des données et du coût de la traversée du système d'information et de gestion de la grille, pour réaliser les interactions graphiques standard (rotation, zoom etc.), et celles spécifiques de la supervision du calcul pour une applicaton particulière.

7.2.3 Problématiques

Nous résumons ici brièvement ce que nous pensons être les principales questions nouvelles par rapport à l'acquis du contexte des grilles de calcul posées par l'intégration d'une contrainte d'interactivité. Certaines sont développées plus longuement dans la section suivante, ainsi que les directions de recherche correspondantes.

1. La définition de la spécificité des services pour l'interactivité, et de leur positionnement par rapport aux standards de facto émergents, ainsi que par rapport aux intergiciels existants, tant dans la communauté française qu'internationale. La veille technologique dans ce domaine est particulièrement difficile : on peut noter qu'il a été jugé nécessaire de créer le projet GridStart pour coordonner les projets de grille du FP5.

2. La prise en compte à chaque niveau des contraintes de l'interactivité dans l'exploration, la recherche, l'accès et le traitement à distance des données, et son intégration dans un formalisme commun. L'exécution efficace d'une requête dépend de sa capacité à effectuer les traitements sur ses données au cours de leur progression depuis la ou les sources dans les délais impartis, et donc de sa capacité à distribuer efficacement tout ou partie de ces traitements en relation avec la localisation initiale et les localisations intermédiaires de ces données. L'interactivité suppose aussi de pouvoir interrompre une requête en cours de traitement et de pouvoir exécuter une nouvelle requête avec une partie seulement des résultats obtenus. Une question cruciale ici est le coût mouvements de données engendré par les choix de co-allocation ou non des étapes. Ces coûts doivent être exposés au niveau de l'environnement qui prend en compte l'interactivité, et peut-être aussi au niveau de l'application. Ceci est un argument pour :
   - une structuration des applications à base de composants qui généralisent le concept de filtre, en exposant les exigences de chaque filtre en terme de transfert et de calcul ;
   - la définition d'un service de gestion du flot de tâche (workflow), qui fournit le support pour traduire les requêtes sous forme de flot de données et de contrôle à travers des services de niveau inférieur (accès aux données, compression, scheduler parallèle dynamique) et pour allouer les ressources nécessaires à ce workflow en exploitant les services d'information du middleware.

   Dans ce domaine, une référence importante est l'environnement DataCutter développé parJ. Saltz [142, 21, 104]. La différence avec le modèle très classique de graphe de tâches n'est pas dans le modèle abstrait de tâches qui communiquent au début et à la fin, mais dans la granularité : on ne s'intéresse plus ici aux nids de boucles (grain fin) du calcul numérique, mais, à des tâches hétérogènes à grain moyen ou gros.

Dans le domaine du calcul numérique, l'analogie serait plutôt avec le placement des calculs minimisant le mouvement des données globales dans des environnements de déport du calcul de type Netsolve ou Diet [44, 8, 39]. A l'extrème, la communication entre tâches s'effectue par fichiers ([41, 185]).

3. Dans le domaine algorithmique, une idée directrice est celle d'*algorithmes à fidélité variable* [139], capables de fournir une approximation de l'information demandée qui sera suffisante pour une partie importante de l'interaction, tout en préservant la possibilité d'une information graduellement plus complète.

   Il y a ici une relation importante avec les travaux menés dans le cadre du programme NSF *Dynamic Data-Driven Application Systems* (DDDAS) [31, 65] : la nécessité de définir un formalisme pour adapter dynamiquement les calculs effectués, au niveau de la résolution comme au niveau des modèles théoriques, en fonction d'une contrainte temporelle, en latence ou en débit, fournie ultimement par une entrée extérieure - plutôt simulation ou capteurs dans le cadre de DDDAS, interaction humaine ici.

   La question est alors la définition d'une méthodologie générale permettant
   - de décrire de façon unifiée les méthodes multi-échelle ou multi-résolution qui doivent être présentes à tous les niveaux ;
   - d'articuler de telles decriptions avec les informations fournies par les outils de surveillance de grilles pour exposer les compromis temps-encombrement contre précision aux niveaux de contrôle du système, incluant l'utilisateur .

   Au niveau applicatif, il s'agit d'intégrer les algorithmes spécifiques du domaine : par exemple le traitement d'images multi-résolution en imagerie médicale [150, 189], ou plus simplement les paramètres d'entrée, en simulation de physique (cf 5.6.2), ou encore le pas de discrétisation en recontruction volumique, doit pouvoir évoluer dynamiquement. Au niveau des services spécifiques, il s'agit par exemple de la compression adaptative.

4. La prise en compte des contraintes de latence sur l'accès aux données. Les méthodes actuellement existantes sur les grilles, à base de techniques de cache (Internet Backplane Protocol - IBP) ou de réplication (DataGrid), n'offrent pas de garantie de performances suffisantes. Un

service essentiel est celui d'algorithmes de compression/décompression [192, 169] adaptatifs [117] : d'une part, les traitements et les communications peuvent se recouvrir ; d'autre part, l'information elle-même peut être structurée pour permettre une représentation hiérarchique adaptable aux capacités de transmission et de traitement.

5. La prise en compte de contraintes de latence sur le traitement des données. On considère ici des traitements parallèles avec placement et répartition de charge dynamiques et supervision interactive. La prise en compte de ce modèle de calcul dans les environnements de grille demande le développement d'ordonnanceurs au niveau applicatif, et d'un schéma généraliste pour leur intégration dans des environnements (grilles) où la soumission de tâches est fortement médiatisée.
6. Le couplage entre exécution locale et exécution sur grilles. Cet aspect est encore très mal pris en compte : la proposition de spécification de négociation de qualité de service élaborée par le Working Group GRAAP du GGF en 2004 (fig. 7.2) reste dans un modèle de production complètement découplé - en fait de type batch - où le transfert d'information entre utilisateur et application s'effectue par fichier.

7.2.4 Technologies

Dans [136], le terme *services interactifs lourds* (interactive hevyweight services) a été proposé pour décrire les composants logiciels spécifiques de l'interactivité sur grille dans le cadre de la supervision du calcul (*computational steering*). Même si le terme de services est actuellement devenu un peu fourre-tout, cette terminologie positionne correctement la spécificité de l'interactivité du point de vue technologique. Le Global Grid Forum construit un ensemble de propositions de standards pour une unification des composants des intergiciels de grille basée sur les Web Services (table 7.1). Une difficulté forte réside dans le fait que les Webs Services instancient un modèle Client-Serveur stateless, alors que les fonctions système de la grille peuvent demander la mémorisation d'un état. Tous les problèmes de synchronisation et de tolérance aux défaillances réapparaissent, ce qui est à l'origine des difficultés du *Globus Toolkit 3* (GT3). Le consortium Globus a entrepris de développer une proposition de standards, le *Web Services Resource Framework*, WS-RF [75], indépendamment du GGF, qui ciblent la gestion de la notification des

A typical application scenario is the request for executing a computing job. A service provider may, as an agreement provider, post an agreement template available to interested requesters. In this scenario, the agreement template defines a list of applications to be executed, and the software execution environment typically specified in a job submission. *Service consumers are given a quality of service guarantee in terms of number of nodes and/or per node memory and storage for a specific time period. Alternatively, the guarantees can be on the completion time*. A service consumer requesting a submitted job must fill in the name of the application to be executed and *its input and output files*. In addition, the service consumer chooses the number of nodes (or any other resource requirements) that the application is guaranteed to execute on. To submit a job, the consumer retrieves the template from the provider, selects the application name, and *provides the URL of the input and output files as well as the details of resource guarantees*. The filled template is sent as an offer to the provider. *The provider decides whether to accept or reject the offered agreement specifying the job*. This may depend on the queue of jobs waiting to be processed and the current allocation of resources. The service provider answers the offer with a confirmation or a fault. In due time, the service provider processes the job and writes the output file to the URL defined in the agreement.

FIG. 7.2 – Citation de la proposition de standard de Service Level Agreement du GGF GRAAP WG [83]. Les passages soulignés le sont par nous

CDDLM	Configuration Description, Deployment, and Lifecycle Management [78]	Descriptions de la configuration des services et de leur déploiement : instanciation, initialisation, démarrage, arrêt, redémarrage
DRMAA	Distributed Resource Management Application API	La spécification d'une API pour la soumission et le contrôle des travaux en direction d'un ou plusieurs systèmes de gestion de ressources distribuées
GRAAP	Grid Resource Allocation Agreement Protocol [83, 82]	Un protocole commun de gestion des ressources permettant la réservation anticipée de ces ressources. Le protocole entre un meta-scheduler et les schedulers locaux est vu comme un service opaque préexistant
JSDL	Job Submission Description Language [73]	La spécification abstraite d'un JSDL, indépendante des langages de réalisation ; un schéma XML correspondant à cette spécification ; les traductions entre le JDL et un ensemble de systèmes de batch d'usage courant
GSA	Grid Scheduling Architecture [85]	Définition d'une architecture de scheduling qui permette la coopération entre des entités de scheduling gérant des ressources de type divers (réseau, logiciel, données, stockage, unités de calcul). La co-allocation est la réservation sont une des activités essentielles ; l'intégration de politiques de scheduling définies par l'utilisateur est aussi considérée .

TAB. 7.1 – Les principaux sous-thèmes d'activité du GGF à l'intérieur du thème *Scheduling and Resource Management (SRM)*. GSA est un Research Group, les autres sont des Working Groups ayant vocation à produire des propositions de standards

evènements, la description des propriétés des ressources et la description des fautes.

Une standardisation des fonctionnalités et des interfaces est toujours souhaitable. En revanche, les questions d'une part des interfaces utilisateur, d'autre part de l'implémentation des services au niveau des intergiciels, reste ouverte. Par exemple, le projet européen GridLab [79] propose une approche des interfaces utilisateur à partir de *portlets*, composants logiciels permettant l'assemblage de portails. Ce type d'approche ne convient pas bien pour des applications avec une interface graphique de haute qualité et interactive. Le compromis performance/portabilité (à travers la conformité aux standards) doit être étudié pour chacun des composants de l'intergiciel qui contribuent à l'interactivité.

7.3 Scheduling pour l'interactivité

7.3.1 Contextes

Architectures de scheduling

Dans le contexte des grilles, on a les spécificités suivantes, qui expriment diverses modalités du problème de passage à l'échelle.

- Informations incomplètes et imprécises sur l'état du système.
- Dynamicité : le taux d'arrivée et de départ est la somme de celui des composants du système, les défaillances, sans être aussi fréquentes de dans les systèmes de Calcul Global et P2P, doivent être considérées comme un évènement normal.
- Très grande hétérogénéité des ressources, en puissance de calcul et en communication. La composante de placement est donc particulièrement importante, pour respecter la localité des données.

Une caractéristique fondamentale pour l'implémentation est la nécessité d'une pile de protocoles. Le scheduling global de la grille doit être implémenté à travers une *hiérarchie* de mécanismes de scheduling : les schedulers noyau et éventuellement utilisateur au niveau du processeur ; les schedulers de grappes comme LSF ou PBS ou de machine parallèle comme LoadLeveler ; au niveau global de la grille, une entité dont le nom et les fonctions sont encore mal fixés [72], *broker*, *meta-scheduler* etc. Les deux premiers composants sont bien connus.

Dans les systèmes actuellement existants, l'alternative pour les meta-schedulers se situe entre d'une part un système centralisé, à réservation comme le Moab Grid Scheduler (anciennement Maui/Siver) [52] ou Community Scheduling Framework [178], très lié à LSF, d'autre part un système à appariement instantané (*matchmaking*) [186], dont les exemples les plus connus sont le matchmaker de Condor et les brokers d'EGEE. La première solution étend le modèle de production des systèmes existants sur grappes, en ciblant un ordonnancement optimal ; la deuxième gère uniquement le placement et décide de l'allocation des applications au fil de leur arrivée, sur la base d'heuristiques très sommaires ; par exemple, le broker d'EDG/LCG-2 place d'abord en fonction de la localité des données, et ne prend en compte la disponibilité des processeurs que s'il y a que secondairement.

Il existe un consensus pour rejeter la solution de type centralisé [72]. On peut y voir deux motivations : la première est la difficulté intrinsèque du passage à l'échelle des politiques de scheduling pertinentes au niveau des grappes (clusters). La seconde motivation est la nécessité d'une structure hiérachique, qui respecte l'organisation matérielle et logicielle du système. Dans les systèmes actuels, même s'ils sont encore très rudimentaires, le broker gère le placement, en prenant en compte les informations spécifiques de la grille (autorisations, localité des données), et délègue aux schedulers de grappe l'ordonnancement temporel.

Qualité de Service

Les systèmes décrits dans la partie précédente ne permettent pas de spécifier une qualité de service ; ils correspondent donc à la vision "best effort" des systèmes de batch classiques. Quelques tentatives ont eu lieu dans le cadre de Globus pour implémenter la réservation [92].

Dans le modèle de production décrit par [83, 82], le point essentiel est l'introduction de la description de la qualité de service comme une des caractéristiques possibles d'une requête d'exécution d'un job. La proposition du GGF cible la réalisation de cette qualité de service par une négociation dynamique entre les entités qui contribuent au scheduling sur grilles. Il s'agit d'apparier les ressources que gère une entité de scheduling avec les requêtes en décrivant des contraintes de qualité de service souples, par exemple une fenêtre temporelle d'exécution plutôt que des valeurs fixes qui correspondraient à la réservation. L'objectif de la proposition est la définition d'un formalisme (*Service Level Agreement*, SLA), permettant de spécifier les pro-

tocoles de négociation relatifs à la QoS. La fonctionnalité et l'implémentation de ces protocoles restent une question ouverte.

Les travaux sur la réalisation de la QoS sur grille sont encore assez peu nombreux. GARA [92] vise à exposer l'information rassemblée par un environnement de surveillance (monitoring) et de prédiction de performance comme NWS [200]. Au niveau des algorithmes de scheduling, des heuristiques sont développées dans un contexte multi-agent, sur la base du raffinement dynamique de solutions initiales basées sur des informations inexactes [140, 151].

7.3.2 Eléments pour une modélisation

Structure de la charge de travail

Une application interactive fournit une borne supérieure du délai d'exécution (*deadline*), ou de débit en calculs élémentaires, que l'environnement de grille doit satisfaire. De ce point de vue, le scheduling des applications interactves sur grille relève du scheduling temps-réel sur multi-processeurs.

Le problème est en fait plus complexe, à cause d'une part de la structure des applications interactives d'une part, et leur coexistence avec des travaux de type batch d'autre part.

- les applications interactives doivent partager les ressources de la grille avec d'autres les applications, qui n'ont pas de contraintes de deadline. Pour simplifier, on désignera dans la suite ces applications comme travaux *longs*.
- Le niveau intra-requête, qui correspond à l'exécution parallèle des composants d'une application interactive, est d'un grain trop fin pour être pris en compte par les mécanismes d'allocation globaux.

Partage des ressources

Le partage des ressources contraint la modélisation du problème de scheduling.

- Les exigences de type temps réel des travaux interactifs doivent être satisfaites.
- Les retards encourus par les tâches longues à cause des travaux interactifs doivent rester bornés.

- Les travaux interactifs ne doivent pas dégrader pas l'utilisation des ressources.
- Les travaux interactifs doivent pouvoir être intégrés avec toutes les politiques existantes régissant les traitements par lots.

A l'exception de la premières, ces demandes ciblent la coexistence entre travaux interactifs et travaux longs sur la grille. La banalisation des ressources, matérielles logicielles et humaines, fédérées dans une grille reste très éloignée de celle d'un ordinateur personnel, dont la puissance de calcul est essentiellement inutilisée. Les organisations participantes à une grille demandent une allocation de ressources mesurable, efficace et équitable. En particulier, les nœuds locaux définissent des structures de files d'attente sophistiquées pour équilibrer les demandes des différents types d'utilisateurs entre elles et vis-à-vis de la maximisation de l'utilisation des ressources.

Une contrainte importante qui en découle est l'impossibilité de supposer les travaux longs susceptibles de préemption au sens fort : leur exécution ne peut pas être interrompue et redémarrée. Bien que techniquement possible, au moins au niveau d'une grappe (Condor [155], Mosix [13]), cette stratégie intrusive n'est pas sociologiquement réaliste. En particulier, la seule granularité où le partage du temps processeur puisse être effectuée est celle, de très bas niveau, du scheduler système sur chaque processeur par l'intermédiaire des priorités processus.

La prise en compte de l'hétérogénéité des requêtes, si elle est nouvelle sur grilles, a été souvent considéré dans d'autres sous-domaines du scheduling. En particulier, de nombreux travaux portent sur la coexistence du best-effort et des applications contraintes dans plusieurs domaines : scheduling processeur, dans le contexte du *Soft Real Time Scheduling* [3, 110, 64, 159] ; routage de paquets sur réseau, avec les options services différenciés (*Differenciated services*) [23] et contrôle d'admission [127, 66] ; serveurs Web [38, 191]

Scheduling utilisateur

Un service d'ordonnancement au niveau applicatif doit compléter les services de scheduling au niveau global. Une analogie au niveau du système d'exploitation est celle de l'alternative entre flots d'exécution (threads) utilisateurs et flots d'exécution système. De la même façon qu'un environnement de threads utilisateurs évite la pénalité des appels au noyau, un scheduler applicatif de grille évitera la traversée de l'ensemble du système d'information de la grille pour chacune des tâches parallèles de courte durée. De façon plus proche

des grilles, le projet AppLeS (Application Level Scheduling)[18, 42, 19] a explicité le concept d'agents de scheduling *centrés sur l'application* (application-centric) et montré sur des exemples provenant de domaines applicatifs variés les gains de performances obtenus par un scheduling dynamique et local [176, 183].

7.3.3 Eléments pour une architecture

D'après ce qui précède, une architecture de scheduling pour l'interactivité doit gérer trois situations de concurrence et de parallélisme distinctes.

- entre jobs interactifs et jobs longs
- entre jobs interactifs
- entre tâches d'un job interactif

Ces trois situations peuvent être gérées de façon relativement indépendante par deux entités (ou services) distinctes de la grille. La première est un *Portail d'interactivité* (PI), qui apparaît au niveau de la grille comme un client ou une application et qui concentre les fonctionnalités liées à l'interactivité : établissement des flux de communication entre le client à l'extérieur de la grille et les ressources de la grille ; gestion des sessions interactives, incluant l'allocation initiale de ressources, la surveillance de l'exécution, et la réallocation dynamique. La seconde est le service *Scheduler Applicatif* (SA).

Le PI soumet des requêtes au service de meta-scheduling existant de la grille, en les étiquetant comme interactives. Le meta-scheduler dialogue avec les schedulers locaux ; certains (ou tous) les schedulers locaux garantissent aux tâches interactives une qualité de service globale en termes d'une part de réservation d'une fraction de la puissance de calcul qu'ils pilotent, d'autre part de priorité : un exemple simple serait un scheduler de grappe qui ne lance qu'un seul job long et qui garantit une priorité égale ou supérieure aux jobs interactifs. D'autres mécanismes au niveau des schedulers locaux sont possibles, le point important est qu'il restent dans le schéma commun de files d'attente, de non préemption forte, et n'impliquent pas de réservation de segments de temps explicites. Le PI gère également la concurrence entre les applications interactives.

Les services pré-existants de la grille (brokers et schedulers locaux) ont ainsi une vision très synthétique des jobs interactifs, qui pourra probablement s'exprimer dans un formalisme de type WS Agreement [83]. La combinaison des brokers et des schedulers locaux fournit essentiellement une disponibilité en bande passante de calcul, qui doit être cependant consultée dyna-

miquement par le PI pour déterminer quelle fraction de la bande passante théoriquement disponible l'est effectivement en fonction de l'état réel de la grille.

La gestion fine de l'interactivité repose sur les services propres fournis par le PI et les SAs. Ces deux services gèrent la concurrence entre jobs interactifs. Ils doivent assurer :

- l'isolation des jobs interactifs entre eux - chaque job interactif se voit allouer une fraction maximale de la bande passante de calcul dédiée à l'interactivité - ;
- le respect des deadlines pour chaque job ;
- le contrôle d'accès : la bandes passante de calcul interactif étant limitée, les requêtes en excès doivent être rejetées ;
- l'optimisation de l'utilisation de la ressource en bande passante de calcul interactif.

Du point de vue algorithmique, le cadre général qui permet de réaliser l'isolation des jobs et le contrôle d'admission est le General Processor Sharing (GPS), qui correspond à un modèle idéal de quantum de temps infinitésimal ; chaque job est caractérisé par un poids w_i, et l'allocation garantit que chaque job reçoit un service qui est fraction proportionnelle à son poids de la ressource : si $S_i(t, t')$ est le service reçu dans l'intervalle $[t, t']$,

$$\frac{S_i(t, t')}{S_j(t, t')} \geq \frac{w_i}{w_j}$$

Dans le domaine du scheduling de threads sur mono et multiprocesseurs, les implémentations [101, 68, 146] sont basées sur les mécanismes de bas niveau (interruption horloge et priorités), cette discrétisation introduisant une erreur.

Ce type d'algorithme a une portée très générale, puisqu'il est également applicable au scheduling de tâches sur processeur et de paquets sur le réseau. Sa généricité provient évidemment de la possibilité de définir les poids en fonction des contraintes spécifiques de l'application. Par exemple, pour des applications périodiques dans le contexte du scheduling temps-réel, chaque job doit être répété sur une période T, et possède une durée d'exécution d ; le poids associé est $w = d/T$.

L'acquis algorithmique considérable de ce modèle rend particulièrement souhaitable de pouvoir transposer aux grilles. La base de cette transposition consiste à considérer que les unités à scheduler sont les jobs, qui offrent une granularité compatible avec les ordres de grandeur de grille. Le point

difficile est la réalisation du partage par allocation de quantas de temps. Pour les travaux longs, comme on l'a vu plus haut, ce n'est pas envisageable. Pour des jobs de type *bag of tasks*, chaque tâche peut être l'analogue d'un quantum. Le service Scheduler Applicatif joue alors un rôle équivalent à celui des mécanismes de bas niveau pour les processeurs, en lançant les tâches d'un job suivant l'ordonnancement défini par l'algorithme de scheduling. Dans un cas idéal, les tâches ont des durées d'exécution identiques et prévisibles, et l'analogie est presque parfaite. Avec des tâches de durées inégales, si elles sont nombreuses et de durées faibles (ce qui est le cas dans l'exemple gPTM3D), la séquence de plusieurs tâche peut réaliser un tel quantum. La qualité de la réalisation de l'ordonnancement ne sera cependant pas parfaite, ce qui implique d'une part de s'orienter vers des algorithmes d'ordonnancement robustes aux perturbations au niveau du PI, d'autre part d'intégrer des mécanismes de lissage (leaky bucket) et de retour d'information (feedback) au niveau du Sheduler Applicatif.

La spécialisation du PS au temps-réel fort (où les deadlines doivent être respectées sans exception), définit une classe d'algorithmes optimaux pour les multiprocesseurs et les applications périodiques : les algorithmes de type P-fair (proportionate fair), où l'ordonnancement garantit que, pour un quantum de temps fini, l'écart entre l'allocation effective et l'allocation idéale (quantum infini) est au maximum d'un quantum ; dans tous les cas l'ordre de priorité des tâches est earliest deadline first, les variations portant sur les méthodes de résolution des ambiguïtés. L'ordonnancement est défini par une analyse statique (*off-line*), qui vérifie que des ressources suffisantes sont disponibles, et qui calcule les instants de démarrage des tâches. Ce schéma complètement statique peut être adapté à la prise en compte de la dynamicité (arrivées et départs de jobs) dans le contexte du Soft Real Time Scheduling, où l'objectif est de minimiser le nombre de deadlines manquées [181, 45, 180]. D'autres algorithmes [101, 68] basés sur le GPS sont plus directement orientés vers la dynamicité ; leur cible initiale est le scheduling monoprocesseur ; leur extension au contexte multiprocesseur soulève des difficultés nouvelles liées à la composante de placement (schedules infaisables [74]) pour lesquelles des heuristiques ont été proposées [45, 180]. Enfin, dans le contexte de la grille, le placement des jobs peut être contraint, en particulier par des considérations de localité des données.

Un autre angle d'attaque possible est l'analogie avec le routage des paquets dans les réseaux. Pour certaines applications, la quantité de travail de calcul à effectuer peut être modulée dynamiquement en fonction des capacités

de l'environnement. La non-exécution de certaines tâches d'un job correspond alors à une perte de paquets. Une première application possible est de substituer à la négociation de bande passante avec le broker les méthodes de test préalable de capacité (*endpoint admission control*) : la capacité pour un noeud de la grille à accepter des tâches interactives est découverte de façon instantanée et complètement dynamique, et n'implique plus la création de files d'attente spécialisées. La pertinence de la prédiction réalisée ainsi est évidemment à valider expérimentalement dans les conditions d'une grille de production. Une seconde application est de transformer le problème de scheduling des jobs sur les noeuds de la grille en un problème de décision sur la réexécution des tâches avortées ou non lancées. Cette formulation plus locale dans le temps et dans l'espace est probablement mieux adaptée au passage à l'échelle. En revanche, les critères de décision doivent intégrer, outre l'information sur l'état de disponibilité de l'environnement, les spécificités l'application qui doit alors définir une mesure de l'importance des tâches.

7.4 Une applications en imagerie médicale : gPTM3D

L'objectif de ce travail est la gridification de PTM3D, l'environnement d'exploration d'images radiologiques développé au LIMSI par A. Osorio et son équipe [148].

7.4.1 PTM3D

Cette section présente brièvement PTM3D, afin de monter l'intérêt et la complexité de son intégration dans une grille biomédicale.

PTM3D est un système de navigation en temps réel, de transfert, d'archivage, de visualisation d'images et de génération de représentations tridimensionnelles à partir de données TDM, IRM PET-scan, ou échographies 3D. Il est implanté sur un ordinateur de type PC standard et est utilisé en ligne dans les centres d'imagerie médicale [148, 173]. Partant d'examens codés en DICOM, le système conduit à la génération assistée par ordinateur, à partir des choix du praticien, d'une segmentation volumique des organes ou des lésions choisis. Ce système comporte les modules suivants : lecture et transfert d'examens DICOM ; visualisation des examens et navigation 3D ;

détection assistée par ordinateur des zones d'intérêt ; reconstruction volumique à métrique réelle ; visualisation réaliste et manipulation de formes tridimensionnelles.

7.4.2 gPTM3D

L'intégration de PTM3D comme frontal de la grille comprend deux étapes. La première correspond au scenario LDRC, et consiste à déporter la reconstruction volumique ; la seconde correspond au scenario RDRC et s'effectuera dans le cadre d'AGIR.

Gridification de la reconstruction volumique

La reconstruction volumique offre une échelle complète de coûts de calcul, du monoprocesseur (tumeurs, calculs) à plusieurs dizaines de processeurs pour des reconstructions du corps complet avec des données issues de scanners multi-résolution utilisées par exemple en planification d'opérations.

En principe, l'exécution parallèle de l'algorithme de reconstruction volumique ne requiert pas une grille aussi fortement que, par exemple, l'analyse d'une base de données d'images distribuée, puisque les données ne sont pas initialement distribuées et que le calcul est du parallélisme trivial. L'intérêt de la grille, par rapport à une grappe locale, réside dans les concepts clé de la grille - service, organisation virtuelle et accès transparent - : un algorithme de reconstruction est un service qui doit être fourni de façon transparente à toute personne autorisée sans que les institutions utilisatrices aient à investir dans une ressource (une grappe) dont elles n'auraient qu'un usage limité. Inversement, le service ne sera utilisé que s'il est intégré dans un outil unique, qui permet de parcourir les images, de mesurer une tumeur ou de reconstruire un organe complexe pour créer une situation de réalité augmentée [148]. Plus spécifiquement à l'interactivité, le processus de reconstruction peut être interrompu et ses données modifiées pendant la reconstruction, par correction des contours.

La reconstruction volumique est typique d'une situation LDRC : les données - la totalité de l'examen - sont locales au front-end ; seuls les contours des coupes et la représentation des volumes reconstruits doivent être échangés avec la grille. Cette propriété favorise fortement le déploiement en pratique clinique : la confidentialité ne soulève pas de difficulté, sauf en cas de reconstruction du visage, puisqu'aucune metadonnée n'est exportée.

Une infrastructure d'interactivité et le déport de la reconstruction ont été réalisés dans le cadre du projet EGEE (cf 7.4.3).

Support d'expérimentations

Au delà de l'expérimentation des services d'interactivité appliqués à la supervision du calcul en LDRC, PTM3D peut évoluer pour accepter un accès à des données distantes. Le standard DICOM détermine la représentation informatique des examens radiologiques, avec une structure hiérarchique complexe incluant des métadonnées. Cette structure est fortement sollicitée interactivement dans la pratique clinique. La problématique est alors inverse, et beaucoup plus difficile : le volume de données est prohibitif pour un accès interactif à distance (plusieurs centaines de MB pour un examen tomodensitométrique).

La méthode envisagée est de concentrer l'utilisation des ressources sur les données importantes pour l'interaction à un instant donné ; les délais liés à l'interaction humaine sont exploités en arrière plan pour raffiner l'information délivrée in fine. Plus précisément, il s'agit d'exploiter des protocoles de compression ou de streaming extensibles qui intègrent les spécificités des données DICOM et/ou des résultats par les algorithmes de traitement de ces données. Dans un schéma de compression extensible, par exemple JPEG 2000 ou MPEG-4, le signal originel est codé en plusieurs niveaux ; le niveau inférieur contient une version rudimentaire du signal, et chaque niveau supérieur raffine cette information ; ces niveaux sont complémentaires au sens où les niveaux inférieurs sont nécessaires pour décoder l'information du niveau supérieur.

Une des spécifités des images médicales réside dans le processus de hiérachisation de l'importance des données. L'information contenue dans les images médicales excède largement ce qu'un utilisateur peut en appréhender à un instant donné : le système perceptif humain est à la fois indispensable pour analyser ces données, et insuffisant pour les traiter simultanément dans leur totalité.

Ceci a deux types conséquences.

- Pour certaines interactions, une information dégradée est suffisante : par exemple, lors de la recherche d'une région d'intérêt (ROI) basée sur des critères anatomiques. Alors que le facteur de compression obtenu en compression (purement 2D) JPEG2000 sans perte est de l'ordre de 2 :1, il peut atteindre 30 :1 avec perte, et garder une qualité acceptable pour une inspection initiale.

- Lorsque la qualité originale de l'information est nécessaire, seul un segment de cette information est nécessaire, en particulier à cause du *fenêtrage radiologique*, qui est la sélection d'une plage dans les niveaux de gris adéquate au type d'organe ou de pathologie visé.

Quelques pistes pour exploiter ces spécificités sont les suivantes.

- Sélection automatique : des mécanismes de préchargement intelligents qui exploitent la façon dont les données sont explorées et extrapolent les caractéristiques de l'exploration.
- Un modèle pour décrire les besoins qui en résultent et les propager vers les sources de données.
- Au niveau le plus proche du matériel, des schémas d'accès optimisés en relation avec la structure d'enregistement sur disque des données DICOM.

La séléction des données doit ensuite être combinée avec des algorithmes généraux ou spécifiques de compression. Si des algorithmes spécifiques sont définis, les problématiques classiques en streaming pourraient être revistées pour intégrer leur spécificité, en particulier l'optimisation de l'utilisation des caches [172, 201], et les protocoles de gestion des paquets au niveau réseau, pour les serveurs d'images médicales sur grilles [115]. La collaboration, dans AGIR, de chercheurs spécialistes de la compression d'une part, des intergiciels de grilles d'autre part, avec des praticiens médicaux et des experts du format DICOM constitue une opportunité exceptionnelle pour la définition d'outils performants et réalistes.

Un problème disjoint, mais également important est la transcription des métadonnées vers des formats compatibles avec les outils standard d'échange et de fouille de données sur le web, ansi que la résolution des problèmes critiques de confidentialité associés à un environnement plus ouvert [128, 111].

7.4.3 Développements

Les développement se sont effectués dans le cadre du projet européen EGEE.

Le projet EGEE

EGEE (Enabling Grids for E-Science in Europe) est un très grand projet de deux ans dans un programme FP6 de quatre ans. Il vise à intégrer des grilles nationales, régionales ou thématiques, de calcul ou de données,

pour créer une infrastructure de grille européenne contribuant à l'European Research Area. Le consortium EGEE implique 70 établissements dans 27 pays, fédérés en grilles régionales, d'une capacité combinée de plus de 20000 CPUs, de loin la plus grande infrastructure internationale de grille à ce jour. Les grilles biomédicales sont un des deux domaines d'applications pilotes d'EGEE, avec une accentuation de l'effort vers l'amélioration de la performance vue par l'utilisateur, en particulier à travers son activité Application Identification and Support, à laquelle je collabore. Un apport essentiel de cette collaboration est de donner accès à un environnement de grille de production à très grande échelle, ce qui d'une part permet un déploiement effectif de gPTM3D, d'autre part offre des possibilités de mesures des comportements des grilles, enfin fournit un point de référence sur les contraintes de la technologie des intergiciels et de passage à l'échelle des applications.

L'intergiciel d'EGEE dérive de celui développé dans le projet FP5 DataGrid. La version actuelle, LCG-2, est proche de celui de DataGrid, avec des améliorations de performances et d'ergonomie. Une restructuration (gLite) permettant de présenter cet intergiciel sous forme de Services Web est en cours, dans la perspective d'une evolution future vers une formulation compatible avec les standards en cours de développement dans le GGF.

Notre contribution

Notre contribution a été de déployer une infrastructure d'interactivité et gPTM3D-reconstruction volumique dans Datagrid [p10, p33, p35] , qui a été ensuite portée sur EGEE.

L'exécution de travaux interactifs demande d'utiliser l'environnement de soumission d'EGEE comme un environnement de *mise en relation* : l'échelle des coûts de la grille n'est pas compatible avec la granularité des tâches PTM3D (quelques secondes à une minute), ni même avec celle d'un job (une reconstruction volumique), qui doit durer quelques minutes.

L'accès à EGEE s'effectue normalement à travers le composant logiciel *User Interface* (UI), qui médiatise la soumission des travaux. Le UI dialogue avec le Broker et le système d'information pour envoyer le travail vers un *Computing Element* (CE), typiquement une grappe, uploader les fichiers d'entrée présents sur le UI ou localiser et éventuellement envoyer au CE les fichiers d'entrée déjà présents dans le système de répliques (replicas) interne. Les fichiers de sortie peuvent être récupérés localement ou conservés dans le système de répliques. Il est possible de demander le placement sur un

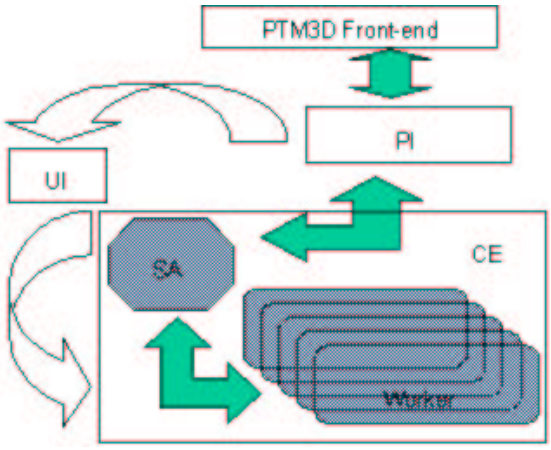


FIG. 7.3 – L'architecture de déport de la reconstruction volumique

CE spécifique ; sinon, l'algorithme de placement a pour premier critère la proximité des fichiers d'entrée, et comme second la charge (en dimension des files d'attente) des CE. EDG/LCG-2 fournit des outils supposés permettre un accès interactif à partir du CE (fig. 7.4 : les flots d'entrée, de sortie et d'erreur standard d'un travail exécuté sur un CE peuvent être redirigés vers des fichiers locaux via un processus fantôme local. Cet outil est un portage dans EGEE de l'outil Bypass [187, 188] développé dans Condor ; sa fonction initiale est le détournement des appels système qu'on ne souhaite pas ou ne peut pas exécuter dans l'environnement distant (par exemple l'ouverture d'un fichier ou la création d'un répertoire local) vers des appels locaux. Cet outil ne fournit qu'un débit très faible, de l'ordre de quelques centaines d'octets par seconde, et ne peut donc pas être utilisé pour établir un canal de communication satisfaisant entre l'UI et les CE pour un flot de données significatif.

La fig. 7.3 donne la structure de l'intégration de PTM3D à travers une architecture d'interactivité. Le lancement d'une session gPTM3D déclenche d'abord une connexion vers un PI, qui relaie les informations entre le monde EGEE et le poste PTM3D, puis la soumission d'un groupe de travaux composés d'un Scheduler Applicatif (SA) et d'un ensemble de Workers. La communication inter-CE n'étant pas actuellement réalisée dans LCG-2, le placement du groupe est limité à un CE. Pour des raisons pratiques, MPI étant rarement disponible sur le CE, la communication entre Workers et Scheduler Applicatif s'effectue via des canaux TCP. Le Scheduler Applicatif est soumis comme un travail LCG-2 interactif, pour initialiser les paramètres de la mise en relation. Le SA établit une connexion avec le PI ; à partir de ce point, le CE et le PI ne communiquent plus que par ce canal. Le canal transporte ensuite les données d'entrées et de sortie des tâches, et le contrôle. Le fonctionnement du PI et du SA sont strictement data-driven : les workers attendent l'arrivée d'une donnée d'entrée (une paire de coupes pour gPTM3D) pour exécuter une unité de calcul (de maillage pour gPTM3D). Le SA gère dynamiquement le tamponnement et l'allocation des données d'entrée aux Workers, pour l'instant en mode FIFO, et retourne les résultats à la volée au PI, qui les relaie au poste PTM3D.

Le développement qui vient d'être décrit founit les composants essentiels pour déployer une architecture d'interactivité complète. Dans sa version actuelle, il a permis le portage de PTM3D sur EGEE avec une accélération linéaire jusqu'aux limites de la granularité des tâches. A court terme, il incluera un mécanisme de retour d'information permettant, par exemple, le

```
JOB SUBMIT OUTCOME The job has been successfully submitted to the
Network
Use edg-job-status command to check job current status.
Your job identifier (edg_jobId) is :
- https ://boszwijn.nikhef.nl :9000/J2gDiZCXARgxKGXbUYeJLA
--- The Interactive Streams have been successfully generated with
the following parameters :
Host : 134.158.88.95
Port : 21256
Shadow process Id : 5489
Input Stream location : /tmp/listener-J2gDiZCXARgxKGXbUYeJLA.in
Output Stream location : /tmp/listener-J2gDiZCXARgxKGXbUYeJLA.out
Error Stream location : /tmp/listener-J2gDiZCXARgxKGXbUYeJLA.err
--- *** Warning *** Make sure you will kill the Shadow process and
remove the input/output streams when interaction finishes
```

FIG. 7.4 – Rapport du lancement d'un travail pseudo-interactif sous EDG

lancement de nouveaux groupes (SA, Workers) lorsque les ressources allouées ne sont pas suffisantes ; plus généralement, un protocole sur le canal Portail d'interactivité/Scheduler Applicatif doit être explicité. A moyen terme, nous visons d'une part à proposer l'infrastructure d'interactivité comme un service autonome au dessus de gLite, d'autre part à expérimenter en vraie grandeur l'intégration d'algorithmes de scheduling comme décrit en 7.3.3.

7.5 Vers un observatoire de la grille

Les infrastructures informatiques à toutes les échelles présentent des caractéristiques des systèmes complexes, au sens technique du terme. A une extrémité, la structure spatio-temporelle des références mémoire d'une exécution séquentielle est fractale ; à l'autre, la connectivité d'Internet a été considérée pendant quelques années comme en loi de puissance, bien que la validité de cette caractérisation soit actuellement remise en question. Au niveau intermédiaire, la disponibilité des machines (postes de travail individuels, serveurs, nœuds de grappes) a fait l'objet de nombreuses études : dans le contexte des systèmes de vol de cycles, il s'agit de définir la loi de probabilité

de la réclamation d'une machine par ses utilisateurs prioritaires, et on trouve alors des distributions heavy-tailed [147, 30] ; dans le contexte du scheduling pour la QoS en exécution sur machines partagées, où l'objet d'étude est la série temporelle de la charge, on trouve des caractéristiques d'auto-similarité et de non-stationarité [63, 160]. En analyse du trafic des réseaux, l'hypothèse poissonienne est considérée depuis longtemps comme trop simplificatrice, et remplacée par des modèles où apparaissent des distributions heavy tailed et des dépendances longues, liées à la superposition de sources on/off , en particulier à partir de l'étude expérimentale du trafic Web [131, 90, 199]. Enfin, la structure des accès dans les systèmes de partage de fichiers à grande échelle a montré des propriétés de type petit monde, qui contribuent d'ailleurs à l'utilisabilité de ces systèmes [113].

Une grille de calcul et de stockage combine l'ensemble des composants qui viennent d'être cités, réseaux et processeurs évidemment, mais aussi accès aux données, sous forme le plus souvent de fichiers, et dont les propriétés de localité sont inexplorées : on ne peut donc envisager de modéliser la grille que comme un système complexe.

La grille constitue ainsi un objet d'étude à un triple point de vue.

- En tant que phénomène, certes artificiel, mais dont les lois de comportement ne sont pas connues.
- En vue d'une modélisation qui fournisse des hypothèses réalistes pour l'étude des systèmes distribués à très grande échelle, et l'algorithmique applicative sur ces systèmes. Le problème du scheduling pour l'interactivité tombe évidemment dans cette catégorie.
- Pour sa gestion en production, en particulier du point de vue du suivi de la consommation des ressources.

Fonder ces recherches renvoie à la nécessité d'études expérimentales qui, pour être fructueuses, doivent se situer à la même échelle que l'objet étudié. La première difficulté de telles études est intrinsèque : la mesure sur les systèmes complexes est une problématique scientifique en elle-même ; un exemple intéressant est la remise en cause actuelle de l'hypothèse de lois d'échelle sur la connectivité d'Internet, qui pourrait provenir d'effets d'horizon sur les sondes expérimentales [47, 126]. La seconde difficulté est pratique. Il existe des environnements permettant de prendre des mesures élémentaires (NWS [200]) ; dans EGEE, des outils de monitoring spécifiques au projet (en particulier test des capacités des dispositifs de stockage de masse) peuvent enregister le comportement du système, charge des machine, débit réseau, structure temporelle, spatiale et d'appartenance, pour l'accès aux fichiers.

Enregistrer des séries temporelles (traces) significatives et utilisables implique des ressources matérielles importantes en stockage et une expertise dans la représentation de données complexes d'une part, l'architecture et la gestion de grandes bases de données d'autre part. Enfin, il faut disposer, non seulement d'un accès à une grille, mais d'une collaboration avec les institutions gestionnaires de la grille qui permette de déployer réellement les instruments de mesure.

Cette perspective sera développée dans le cadre d'un programme pluriformations en cours d'évaluation.

Annexe A

Document

Annexe B

Liste de publications

Ouvrages et chapitres

[p1] C. Germain et J. Sansonnet. *Les ordinateurs massivement parallèles.* Armand-Colin, 1991. Traduit en Espagnol : *Ordenadores Masivamente Parallelos.* Paraninfo, 1993.

[p2] F. Cappello, A. Djilali, G. Fedak, C. Germain, O. Lodygensky et V. Neri. *Calcul réparti a grande échelle*, chapitre XtremWeb : une plateforme de recherche sur le calcul global et pair a pair, pages 153–186. Lavoisier, 2002.

Thèse

[p3] *Etude des mécanismes de communication pour une machine massivement parallèle : MEGA.* Thèse de l'Université Paris-Sud, direction D. Etiemble.

Articles

[p4] C. Germain, J. Béchennec, D. Etiemble et J. Sansonnet. A communication architecture for a massively message-passing multicomputer. *Jal. Parallel and Distibuted Computing*, 19 :338–348, 93

[p5] C. Germain, F. Delaplace et R. Carlier. A static execution model for data-parallelism. *Parallel Processing Letters*, 4(4) :367-378, 1994.

[p6] F. Cappello et C. Germain. The static network : A high performance reconfigurable communication network. *Parallel Processing Letters*, 5(1) :97-109, 1995.

[p7] D. Gautier de Lahaut et C. Germain. A static approach for compiling communications in parallel scientific programs. *Scientific Programming*, 4, 1995.

[p8] C. Germain et V. Néri. Java-based coupling for data parallel predictive-adaptive domain decomposition. *Scientific Programming*, 7(2) :185-189, 99.

[p9] T. Brandes et C. Germain. A schedule cache for data-parallel unstructured computations. *Parallel Computing*, 26(13) :1807-1823, 2000.

[p10] C. Germain, R. Texier et A. Osorio. Exploration of Medical Images on the Grid. *Methods of Information in Medecine*, 2005. To appear.

Conférences

[p11] C. Germain, J. Béchennec, D. Etiemble et J-P. Sansonnet. An interconnection network and a routing scheme for a massively parallel message passing multicomputer. In *3rd Symp. on Frontiers of Massively Parallel Computation*, College Park, MD, 1990.

[p12] J. Béchennec, C. Germain et V. Neri. An efficient hardwired router for a trimensionnal mesh interconnection network. In *CompEuro91*, Bologne, Italie, 1991.

[p13] J. Giavitto, C. Germain et J. Fowler. OAL : an implementation of an actor language on a massively parallel message-passing architecture. In *2nd European Distributed Memory Computing Conf.*, Munich, Allemagne, 1991.

[p14] F. Cappello, J. Béchennec, D. Delaplace, C. Germain, J. Giavitto, V. Neri, et D. Etiemble. A parallel architecture based on compiled communication schemes. In *ParCo93*, pages 371-378. Elsevier, 1993.

[p15] F. Delaplace et C. Germain. Test d'identification des communications statiques. In *RenPar 5*, Brest, France, 1993.

[p16] F. Cappello, J. Béchennec, F. Delaplace, C. Germain, J. Giavitto, V. Neri, et D. Etiemble. Balanced distributed memory parallel computers. In *Int. Conf. on Parallel Processing 93*, pages 72-76. CRC Press, 1993.

[p17] F. Cappello, F. Delaplace et C. Germain. Static communications for efficient massive parallelism. In *3rd Int. Workshop on Interconnection Networks*, 1993.

[p18] F. Cappello et C. Germain. Toward high communication performance through compiled communications on a circuit switched interconnection network. In *1st IEEE Symp. on High Performance Computer Architecture*, pages 44-53. IEEE, 1995.

[p19] D. Etiemble et C. Germain. Standard microprocessors versus custom processing elements for massively parallel architectures. In *PaCT95*, LNCS 964, pages 320-324. Springer Verlag, 1995.

[p20] D. Gautier de Lahaut et C. Germain. Static communications in parallel scientific programs. In *Parle94*, LNCS 817, pages 262-276. Springer Verlag, 1994.

[p21] C. Germain et F. Delaplace. Automatic vectorization of communications for data-parallel programs. In *Europar95*, LNCS 966, pages 429–440. Springer Verlag, 1995.

[p22] F.Coelho, C. Germain et J.-L. Pazat. Compiling HPF. In G.R. Perrin, editor, *The Data Parallel Programming Model*, LNCS 1132, pages 104–130. Springer Verlag, 1996.

[p23] C. Germain et D. G. de Lahaut. Improving irregular parallel communication through sorting. In *HPCN'97*, LNCS 1225, pages 820–829. Springer, 97.

[p24] C. Germain et F. Delaplace. Compiling the block-cyclic distribution. In *PARCO'97*. Elsevier, 97.

[p25] C. Germain, J.Laminie, M. Pallud et D. Etiemble. An HPF case study of a domain-decomposition based irregular application. In *PaCT'97*. LNCS 1277, pages 201-209, 97.

[p26] T. Brandes et C. Germain. A tracing protocol for optimizing data parallel irregular communications. In *EUROPAR'98*, LNCS 1470, pages 629-638. Springer, 98.

[p27] C. Germain, V. Néri, G. Fedak et F. Cappello. Xtremweb : Building an experimental platform for global computing. In *Proc. 1st IEEE ACM Intl. Workshop Grid 2000*. Springer, 2000.

[p28] G. Fedak, C. Germain, V. Neri et F. Cappello. XtremWeb : A generic global computing platform. In *IEEE/ACM CCGRID'2001*, pages 582-587, IEEE Press, 2001.

[p29] C. Germain, G. Fedak, V. Neri et F. Cappello. Global Computing Systems. In *3rd Int. Conf. on Large Scale Scientific Computations*, LNCS 2179, pages 218–227, Sozopol, 2001. Springer-Verlag.

[p30] A. Selikhov, G. Bosilca, C. Germain, G. Fedak et F. Cappello. MPICH-CM : a Communication Library Design for a P2P MPI Implementation. In *9th Euro PVM/MPI Conf.*, LNCS 2474, pages 323–330, Vienna, Oct. 2002. Springer-Verlag.

[p31] G. Bosilca, A. Bouteiller, F. Cappello, S. Djilali, G. Fedak, C. Germain, T. Hérault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri et A. Selikhov. MPICH-V : Parallel Computing on P2P Systems . In *IEEE/ACM Int. Conf. for High Performance Computing and Communications 2002 (SC'02 - SuperComputing'02)*, Baltimore, 2002.

[p32] A. Selikhov and C. Germain. CMDE : a Channel Memory based Dynamic Environment for Message Passing based on MPICH-V Architecture. In *Procs. Parallel Computing Technologies Conf.(PaCT)*, LNCS 2763, pages 528–537, Novosibirsk, Sept. 2003. Springer-Verlag.

[p33] C. Germain, A. Osorio et R. Texier. A Case Study in Medical Imaging and the Grid. In S. Norager, editor, *Procs. 1st European HealthGrid Conference*, pages 110–118, Lyon, Jan. 2003. EC-IST.

[p34] C. Germain and N. Playez. Result-Checking in Global Computing Systems. In *Procs.17h ACM Int.Conf. on Supercomputing*, pages 226–233, San Francisco, June 2003. ACM Press.

[p35] C. Germain, R. Texier et A. Osorio. Interactive Exploration of Medical Images on the Grid. In *Procs. 2nd european HealthGrid Conference*, Clermont-Ferrand, Jan. 2004.

[p36] C. Germain-Renaud et D. Monnier. Grid Result-Checking. Soumis à *Computing Frontiers.*

[p37] C. Germain, V. Breton, P. Clarysse, Y. Gaudeau, T. Glatard, E. Jeannot, Y. Legré, C. Loomis, J. Montagnat, J-M Moureaux, A. Osorio, X. Pennec, and R. Texier. *Grid-enabling medical image analysis.* 2005. soumis à BioGrid 2005.

Autres

[p38] D. Berry, C. Germain-Renaud, D. Hill, S. Pieper et J. Saltz. Report on workshop image 03 : Images, medical analysis and grid environments.

TR UKeS-2004-02, UK National e-Science Centre,
http ://www.nesc.ac.uk/technical_papers/UKeS-2004-02.pdf, Feb. 2004.

[p39] A. Osorio, S. Merran, C. Germain, J. Atif, X. Ripoche, A. Tarault et R. Texier. Segmentation 3D d'images radiologiques : application à la reconstruction et mesure du volume d'organes et de lésions. *Journal de Radiologie* (recueil de résumés), 84(10), 2003. Journées Françaises de Radiologie.

Bibliographie

[1] *Embracing Failure : A Case for Recovery-Oriented Computing (ROC)*, 2001.

[2] I. Foster et al. A. L. Chervenak. Giggle : A Framework for Constructing Scalable Replica Location Services. In *SC2002*, 2002.

[3] Luca Abeni and Giorgio Buttazzo. Adaptive bandwidth reservation for multimedia computing. In *Procs 6th Int. Conf. on Real-Time Computing Systems and Applications*. IEEE Computer Society, 1999.

[4] Lada A. Adamic, Rajan M. Lukose, Amit R. Puniyani, and Bernardo A. Huberman. Search in power-law networks. *Phys. Rev. E*, 64(4) :046135, 2001.

[5] D. P. Anderson. BOINC : A System for Public-Resource Computing and Storage. IEEE Computer Society Press, 2004.

[6] T. Anderson, D. Culler, and D. Patterson. A Case for NOW (Networks of Workstations). *IEEE Micro*, 15(1) :54–64, 1995.

[7] Gabriel Antoniu, Luc Bougé, , and Mathieu Jan. JuxMem : Weaving together the P2P and DSM paradigms to enable a Grid Data-sharing Service. *Journal of Supercomputing*, 2004.

[8] D. Arnold, H. Casanova, and J.Dongarra. Innovations of the Netsolve Grid Computing System. *Concurrency : Practice and Experience*, 14(13-15) :1457–1479, 2002.

[9] Satoshi Asami, Nisha Talagala, and David A. Patterson. Designing a Self-Maintaining Storage System. In H. Jin, T. Cortes, and R. Buyya, editors, *High Performance Mass Storage and Parallel I/O : Technologies and Applications*, pages 453–463. IEEE Computer Society Press, 2001.

[10] Obervatoire Pierre Auger. http ://www.auger.org/.

[11] J. E. Baldeschwieler, R. D. Blumofe, and E. A. Brewer. Atlas : An Infrastructure for Global Computing . In *7th ACM SIGOPS*, 96.

[12] A. Barak, S. Guday, and R. G. Wheeler. The MOSIX distributed operating system : load balancing for UNIX. volume 672 of *LNCS*, page 221, New York, NY, USA, 93. Springer-Verlag.

[13] A. Barak and O. La'adan. Performance of the MOSIX Parallel System for a Cluster of PCs. In *Procs Int. Conf. on High-Performance Computing and Networking*, pages 624–635, 1997. LNCS 1225.

[14] V. Barnett and T. Lewis. *Outliers in Statistical Data*. John Wiley and Sons, 84.

[15] A. Bassi, M. Beck, T. Moore, J.S. Plank, M. Swany, R. Wolski, and G. Fagg. The Internet Backplane Protocol : a study in resource sharing. *Future Gener. Comput. Syst.*, 19(4) :551–562, 2003.

[16] O. Beaumont, A. Legrand, and Y. Robert. Scheduling divisible workloads on heterogeneous platforms. *Parallel Comput.*, 29(9) :1121–1152, 2003.

[17] J. Beberg. The Cosm Project. http ://www.mithral.com/projects/cosm/, 2003.

[18] F. Berman. *High-Performance Scheduling*, chapter 12. Morgan Kaufmann, 99.

[19] F. Berman, R. Wolski, and H. Casanova et al. Adaptive computing on the grid using apples. *IEEE Trans. Parallel Distrib. Syst.*, 14(4) :369–382, 2003.

[20] D.P. Bertsekas and J.N. Tsitsiklis. Some aspects of parallel and distributed iterative algorithms - a survey. *Automatica*, 27 :3–21, 1991.

[21] M. Beynon, T. Kurc, A. Sussman, and J. Saltz. Design of a Framework for Data-Intensive Wide-Area Applications. In *9th Heterogeneous Computing Workshop*, 2000.

[22] S. N. Bhatt, F. Chung, F. T. Leighton, and A. L. Rosenberg. On Optimal Strategies for Cycle-Stealing in Networks of Workstations. *IEEE Trans. on Computers*, 46(5) :545–557, 97.

[23] S. Blake, D. Black, , M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differenciated Services, 1998. IETF RFC 2475.

[24] M. Blum and S. Kannan. Designing programs that check their work. In *21st ACM Symposium on the Theory of Computing (STOC)*, pages 86–97, 1989.

[25] M. Blum and H. Wasserman. Software reliability via run-time result-checking. *Journal of the ACM*, 44(6) :826–849, 97.

[26] A. Borodin and J.E. Hopcroft. Routing, merging and sorting on parallel models of computation. *Jal. of Computer and System Science*, (30) :130–145, 1985.

[27] G. Bosilca, A. Bouteiller, F. Cappello, S. Djilali, G. Fedak, C. Germain, T. Herault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri, and A. Selikhov. MPICH-V : Parallel Computing on P2P Systems. In *IEEE-ACM SC'2002*, 2002.

[28] A. Bouteiller, F. Cappello, T. Herault, G. Krawezik, P. Lemarinier, and F. Magniette. MPICH-V2 : a fault tolerant MPI for volatile nodes based on pessimistic sender based message logging. In *IEEE-ACM SC'2003*, 2003.

[29] A. Bouteiller, P. Lemarinier, G. Krawezik, and F. Cappello. Coordinated checkpoint versus message log for fault tolerant mpi. In *IEEE Cluster 2003*, pages 242–250, 2003.

[30] J. Brevik, D. Nurmi, and R. Wolski. Automatic Methods for Predicting Machine Availability in Desktop Grid and Peer-to-Peer Systems. In *IEEE/ACM Int. Symp. on Cluster Computing and the Grid*, pages 261–276. ACM Press, 2004.

[31] M. Bubak, G. D. van Albada, P. M. A. Sloot, and J.J. Dongarra (eds.). *Workshop on Dynamic Data-Driven Application Systems*, volume 3038 of *LNCS*. Springer-Verlag, 2004. ICCS 2004.

[32] P. Buncic, A. J. Peters, and P.Saiz. The AliEn system, status and perspectives. In *Computing in High Energy and Nuclear Physics*. econf - http ://www.slac.stanford.edu/econf/C0303241, 2003.

[33] J-M Busca, M. Bertier, F. Belkouch, P. Sens, and L. Arantes. A Performance Evaluation of a Quorum-Based State-Machine Replication Algorithm for Computing Grids. In *Procs. 16th Symposium on Comp uter Architecture and High Performance Computing (SBAC-PAD '04)*. IEEE Comp. Soc., 2004.

[34] C. Calgaro, A. Debussche, and J. Laminie. On a multilevel approach for the two dimensional navier-stokes equations with finite elements,. *Int. J. for Num. Meth. in Fluids*, 27 :241–258, 1998.

[35] G. Candea, J. Cutler, and A. Fox. Improving availability with recursive microreboots : a soft-state system case study. *Perform. Eval.*, 56(1-4) :213–248, 2004.

[36] G. Candea and A. Fox. Crash-only software. In *Procs of the 9th Workshop on Hot Topics in Operating Systems*, 2003.

[37] F. Cappello. Modèles d'exécution parallèles pour les architectures hautes performances et les grands systèmes distribués. Habilitation à diriger des recherches. Université Paris-Sud, 2001.

[38] J. Carlstrom and R. Rom. Application-aware Admission Control and Scheduling in Web Servers. In *IEEE Infocom 2002 Conf.*, June 2002.

[39] E. Caron, F. Desprez, M. Quinson, and F. Suter. Performance Evaluation of Linear Algebra Routines. *Parallel Computing*, 2004. Special issue on Clusters and Computational Grids for Scientific Computing (CCGSC'02).

[40] Enrique V. Carrera and Ricardo Bianchini. Efficiency vs. portability in cluster-based network servers. In *Procs of the 8th ACM SIGPLAN symp. on Principles and practices of parallel programming*, pages 113–122. ACM Press, 2001.

[41] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. In *9th Heterogeneous Computing Workshop*, pages 349–363, 2000.

[42] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS parameter sweep template : user-level middleware for the grid. In *Procs 2000 ACM/IEEE conference on Supercomputing (CDROM)*, 2000.

[43] H. Casanova, M.G. Thomason, and J.Dongarra. Stochastic performance prediction for iterative algorithms in distributed environments. *J. Parallel Distrib. Comput.*, 58(1) :68–91, 1999.

[44] Henri Casanova and Jack Dongarra. NetSolve : A network-enabled server for solving computational science problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3) :212–223, 1997.

[45] A. Chandra, M. Adler, and P. Shenoy. Deadline fair scheduling : Bridging the theory and practice of proportionate-fair scheduling in multiprocessor servers. In *Proc. of the 7th IEEE Real-Time Technology and Applications Symposium*, 2001.

[46] K.M. Chandy and L. Lamport. Distributed snapshots : determining global states of distributed systems. *Trans. Computer Systems*, 3(1) :63–75, 1985.

[47] H. Chang, R. Govindan, S. Jamin, S. J. Shenker, and W. Willinger. Towards capturing representative as-level internet topologies. *Computer Networks*, 44(6), 2004.

[48] D. Chazan and W. L. Miranker. Chaotic relaxation. *jal Linear Algerbra and Applications*, 2 :199–222, 1969.

[49] B.S. Chlebus, R. De Prisco, and A. A. Shvartsman. Performing tasks on synchronous restartable message-passing processors. *Distributed Comput*, 14 :49–64, 2001.

[50] D. D. Clark. The design philosophy of the DARPA internet protocols. In *SIGCOMM*, pages 106–114. ACM Press, 1988.

[51] E.G. Coffman, L. Flatto, and A. Y. Kreinin. Scheduling saves in fault-tolerant computations. *Acta Informatica*, 30 :409–423, 93.

[52] Maui Consortium. Moab Grid Scheduler Overview. http ://www.clusterresources.com/products/mgs/docs/silveroverview.shtml.

[53] G. Da Costa and O. Richard. Impact of realistic workload in peer-to-peer systems a case study : Freenet. In *International Symposium on Parallel and Distributed Computing (ISPCD)*, 2002.

[54] L. Cottrell. Monitoring Internet Connectivity of Educational and Research Institutions. http ://www.ppdg.net/mtgs/Troubleshooting/ictp-oct02.ppt, 2002.

[55] Russ Cox, Athicha Muthitacharoen, and Robert Morris. Serving DNS Using a Peer-to-Peer Lookup Service. In *1st Int. Workshop on Peer-to-Peer Systems*, number 2429 in LNCS, pages 155–165. Springer-Verlag, 2002.

[56] O. Cozette, C. Randriamaro, and G. Utard. Read2 : Put disks at network level. In *Procs 3st International Symposium on Cluster Computing and the Grid*. IEEE Computer Society, 2003.

[57] J. Cutler, A. Fox, and K. Bhasin. Applying the Lessons of Internet Services to Space Systems. In *7th IEEE Aerospace Conference*, pages 3287–3295. IEEE Society, 2002.

[58] H. Dail, F. Berman, and H. Casanova. A decoupled scheduling approach for grid application development environments. *J. Parallel Distrib. Comput.*, 63(5) :505–524, 2003.

[59] T. DeFanti, I. Foster, M. Papka, R. Stevens, and T. Kuhfuss. Overview of the I-WAY : Wide area visual supercomputing. *Intl J. Supercomputer Applications*, 10(2-3) :123–131, 1996.

[60] A. Denis, C. Pérez, T. Priol, and A. Ribes. Padico : A component-based software infrastructure for grid computing. In *7th Intl Parallel and Distributed Processing Symp. (IPDPS 2003)*, pages 161–172. IEEE Computer Society, 2003.

[61] P. A. Dinda. The statistical properties of host load. *Scientific Programming*, 7(3,4), 1999.

[62] P. A. Dinda. Online prediction of the running time of tasks. In *Procs of the 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 336–337. ACM Press, 2001.

[63] Peter A. Dinda and David R. O'Hallaron. The statistical properties of host load. 1998.

[64] H. Domjan and T. R. Gross. Managing Resource Reservations and Admission Control for Adaptive Applications. In *Procs Int. Conf. Parallel Processing (ICPP '01)*. IEEE Computer Society, 2001.

[65] C.C. Douglas and F. Darema. Dynamic Data-Driven Application Simulation. http ://www.dddas.org/iccs2004.html.

[66] Z. Duan, Z. Zhang, Y. T. Hou, and L. Gao. A Core Stateless Bandwidth Broker Architecture for Scalable Support of Guaranteed Services . *IEEE Trans. Parallel and Distributed Systems*, 15(2) :167–182, 2004.

[67] B. Ducourthial and S. Tixeuil. Self-stabilization with path algebra. *Theor. Comput. Sci.*, 293(1) :219–236, 2003.

[68] K. J. Duda and D. R. Cheriton. Borrowed-virtual-time (BVT) scheduling : supporting latency-sensitive threads in a general-purpose scheduler. In *Procs 17th ACM Symp. on Operating Systems Principles*, pages 261–276. ACM Press, 1999.

[69] C. Dwork, J. Y. Halpern, and O. Waarts. Performing work efficiently in the presence of faults. *SIAM J. Comput.*, 27(5) :1457–1491, 1998.

[70] E. N. (Mootaz) Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3) :375–408, 2002.

[71] D.H.J. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne. A worldwide flock of Condors : Load sharing among workstation clusters. *Future Generation Computer Systems*, 12 :53–65, 1996.

[72] A. Andrieux et al. Open Issues in Grid Scheduling. Technical Report UKeS-2004-03, UK National e-Science Centre, 2004.

[73] A. Anjomshoaa et al. The JSDL Specification - draft-ggf-jsdl-spec-0.5.2, Sep. 2004. https ://forge.gridforum.org/projects/jsdl-wg/document/draft-ggf-jsdl-spec/en/10.

[74] A. Chandra et al. Surplus fair scheduling : A Proportional-Share CPU scheduling algorithm for symmetric multiprocessors. pages 45–58, 2000.

[75] C. Czajkowski et al. The WS-Resource Framework, May 2004. http ://www.globus.org/wsrf/specs/ws-wsrf.pdf.

[76] D. Düllmann et al. Models for Replica Synchronisation and Consistency in a Data Grid. In *Proceedings of the 10th IEEE Int. Symp. on High Performance Distributed Computing (HPDC-10'01)*, pages 67–76. IEEE Comp. Soc., 2001.

[77] D. E. Culler et al. LogP : Towards a realistic model of parallel computation. In *ACM SIGPLAN Symp. Principles Practice of Parallel Programming*, pages 1–12, 1993.

[78] D.Milojicic et al. CDDLM Foundation Document, Feb. 2004. https ://forge.gridforum.org/projects/cddlm-wg/document/CDDLM_Foundation_Document_v12/en/1.

[79] G. Allen et al. Enabling applications on the grid : A gridlab overview. *Int. Jal. of High Performance Computing Applications*, Aug. 2003.

[80] I. Stoica et al. Chord : a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1) :17–32, 2003.

[81] J. Kubiatowicz et al. OceanStore : An Architecture for Global-scale Persistent Storage. In *Procs 9th ACM ASPLOS*, 2000.

[82] J. Pruyne et al. The WS-AgreementNegotiation Specification, Apr. 2004. https ://forge.gridforum.org/projects/graap-wg/document/WS-AgreementNegotiationSpecificationDraft.doc/en/1.

[83] J. Pruyne et al. WS-Agreement specification draft, June 2004. https ://forge.gridforum.org/projects/graap-wg/document/draft-ggf-graap-agreement-0.9.pdf/en/1.

[84] M. Leech et al. Socks protocol version 5. http ://www.ietf.org/rfc/rfc1928, March 1996. IETF RFC 1928.

[85] P. Wieder et al. Grid Scheduling Use Cases V1.0, July 2004. https ://forge.gridforum.org/projects/document/Grid_Scheduling_Use_Cases/en/1.

[86] S. Tuecke et al. Open Grid Services Infrastructure (OGSI) Version 1.0. http ://www-unix.globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf, 2003.

[87] G. E. Fagg and J. Dongarra. FT-MPI : Fault Tolerant MPI, Supporting Dynamic Applications in a Dynamic World. In *7th European PVM/MPI Users' Group Meeting*, pages 346–353. Springer-Verlag, 2000.

[88] B. Falissard. *Les analyses intermédiaires dans les essais thérapeutiques.* PhD thesis, Université Paris-Sud, 90.

[89] G. Fedak. *XtremWeb : une plate-forme pour l'étude expérimentale du calcul global pair-à-pair.* PhD thesis, Université Paris-Sud, 2003.

[90] A. Feldmann, A. C. Gilbert, W. Willinger, and T. G. Kurtz. The changing nature of network traffic : scaling phenomena. *SIGCOMM Comput. Commun. Rev.*, 28(2) :5–29, 1998.

[91] F. Fellini. Ladri di biciclette, 1949.

[92] I. Foster, M. Fidler, A. Roy, V. Sander, and L. Winkler. End-to-End Quality of Service for High-end Applications. *Computer Communications*, 27(14) :1375–1388, 2004.

[93] I. Foster and A. Iamnitchi. On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. In *2nd Intl Workshop on Peer-to-Peer Systems (IPTPS'03)*, volume 2573 of *LNCS*, pages 118–128. Springer-Verlag, 2003.

[94] I. Foster and C. Kesselman. *Computational Grids*, chapter 2. Morgan Kaufmann.

[95] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid Services for Distributed System Integration. *Computer*, 35(6) :37–46, 2002. extended version : The Physiology of the Grid : An Open Grid Services Architecture for Distributed Systems Integration, GGF June 2002.

[96] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid : Enabling scalable virtual organizations. *Intl Jal Supercomputer Applications,*, 15(3) :200–222, 2001.

[97] I. Foster, H. Kishimoto, and J. Nick. Open grid services architecture working group. https ://forge.gridforum.org/projects/ogsa-wg.

[98] A. Geist, J. A. Kohl, P. M. Papadopoulos, and S. Scott. Beyond PVM 3.4 : What We've Learned, What's New, and Why. In M. Bubak, J. Dongarra, and J. Wasniewski, editors, *Procs.4th European PVM/MPI Users' Group Meeting*, volume 1332 of *LNCS*, pages 116–126. Springer, 1997.

[99] C. Gkantsidis, M. Mihail, and A. Saberi. On the random walk method in peer-to-peer networks. In *INFOCOM 04*, 2004.

[100] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Jal. ACM*, 45(4) :653–750, 1998.

[101] P. Goyal, X. Guo, and H. M. Vin. A Hierarchical CPU Scheduler for Multimedia Operating Systems. In *Usenix Association Second Symposium on Operating Systems Design and Implementation (OSDI)*, pages 107–121, 1996.

[102] A. Grimshaw and al. The legion vision of a worldwide virtual computer. *Comm. ACM*, 15 :607–621, 97.

[103] Trusted Computing Group. https ://www.trustedcomputinggroup.org /downloads/specifications/.

[104] S. Hastings, M. Kurc, S. Langella, U. V. Catalyurek, T. C Pan, and J. H. Saltz. Image Processing for the Grid : A Toolkit for Building Grid-enabled Image Processing Applications. In *Proc. 3rd Int. Symp. on Cluster Computing and the Grid*, pages 36–43, 2003.

[105] D. Heck and al. Fzka, Forschungszentrum Karlsruhe, 98.

[106] E. Heymann, M. A. Senar, E. Luque, and M. Livny. Adaptive Scheduling for Master-Worker Applications on the Computational Grid. In *Procs 1st IEEE/ACM Intl Workshop on Grid Computing (GRID 2000)*, 2000.

[107] J. H. Hine and P. Dagger. Securing distributed computing against the hostile host. In *Procs of the 27th conference on Australasian computer science*, pages 279–286, 2004.

[108] MPI home page. http ://www-unix.mcs.anl.gov/mpi/.

[109] H.Stockinger, A. Samar, K. Holtman, B. Allcock, I. Foster, and B.Tierney. File and Object Replication in Data Grids. *Cluster Computing*, 5(3) :305–314, 2002.

[110] Hao hua Chu. *CPU Service Classes :A Soft Real Time Framework for Multimedia Applications.* PhD thesis, University of Illinois at Urbana-Champaign, 1999.

[111] R. Hussein, U. Engelmann, A. Schroeter, and H.P. Meinzer. Dicom structured reporting. *Radiographics*, 24(3) :891–909, 2004.

[112] A. Iamnitchi and I.T. Foster. A Problem-Specific Fault-Tolerance Mechanism for Asynchronous, Distributed Systems. In *International Conference on Parallel Processing*, pages 4–14, 2000.

[113] A. Iamnitchi, M. Ripeanu, and I. T. Foster. Locating Data in (Small-World ?) Peer-to-Peer Scientific Collaborations. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 232–241, 2002. LNCS.

[114] I.Foster, J. Insley, G. Laszewski, C. Kesselman, and M.Thiebaux. Distance Visualization : Data Exploration on the Grid. *Computer*, 32(12) :36–43, 1999.

[115] I. E. Magnin J. Montagnat, V. Breton. Partitioning medical image databases for content-based queries on a grid. *Methods of Information in Medecine*, 2005. To appear.

[116] S. Jafar, S. Varrette, and J-L. Roch. Using data-flow analysis for resilience and result checking in peer-to-peer computations. In *IEEE DEXA'2004 - Workshop GLOBE'04 : Grid and Peer-to-Peer Computing Impacts on Large Scale Heterogeneous Distributed Database Systems*, 04.

[117] E. Jeannot, B. Knutsson, and M. Bjorkman. Adaptive Online Data Compression. In *11th IEEE Int. Symp. on High Performance Distributed Computing - HPDC 11*, pages 379 – 388, July 2002.

[118] R. Kalawsky. User interaction - Considerations for Improving Computational Grid Environments. 2nd Reality Grid Workshop, June 2003.

[119] G. Kan. *Gnutella*, chapter 8. O'Reilly, 01.

[120] D. Kondo, A. Chien, and H. Casanova. Resource Management for Rapid Application Turnaround. In *SC'O4*. IEEE CS.

[121] D. Kondo, M. Taufer, C. Brooks, H. Casanova, and A. Chien. Characterizing and Evaluating Desktop Grids : An Empirical Study. In *Procs 18th Intl Parallel and Distributed Processing Symposium (IPDPS'04).* IEEE Computer Society Press, 2004.

[122] D. Kondo, E. Wing, H. Casanova, and F. Berman. Models and Scheduling Mechanisms for Global Computing Applications. In *Procs 16th Intl Parallel and Distributed Processing Symposium (IPDPS'02*. IEEE Computer Society Press, 2002.

[123] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Lebofsky. SETI@home-Massively Distributed Computing for SETI. *Computing in Science and Engineering*, 3(1) :78–83, 2001.

[124] K. Krauter, R. Buyya, and M. Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Softw. Pract. Exper.*, 32(2) :135–164, 2002.

[125] I. Kuz, M. van Steen, , and H. Sips. The Globe Infrastructure Directory Service. *Computer Communications*, 25(9) :835–845, 2002.

[126] A. Lakhina, J. Byers, M. Crovella, and P. Xie. Sampling biases in ip topology measurements. In *Procs of IEEE Infocom*, 2003.

[127] L.Breslau, E. W. Knightly, S.Shenker, I.Stoica, and H.Zhang. Endpoint Admission Control : Architectural Issues and Performance. In *SIGCOMM*, pages 57–69, 2000.

[128] K. P. Lee and J. Hu. Xml schema representation of dicom structured reporting. *Jal American Medical Informatics Association (JAMIA)*, 10(2) :213–223, 2003.

[129] N. Leibowitz, M. Ripeanu, and A. Wierzbicki. Deconstructing the Kazaa network. In *3rd IEEE Workshop on Internet Applications (WIAPP'03).*, 2003.

[130] T. Leighton. The challenges of delivering content on the internet. In *Procs of the 20th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems.* ACM Press, 2001.

[131] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic. *IEEE/ACM Trans. Netw.*, 2(1) :1–15, 1994.

[132] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Trans. Netw.*, 2(1) :1–15, 1994.

[133] Jian Liang, Rakesh Kumar, and Keith Ross. Understanding KaZaA. Tr, NY Polytechnic University, 2004.

[134] David Liben-Nowell, Hari Balakrishnan, and David Karger. Observations on the Dynamic Evolution of Peer-to-Peer Networks. In *1st Int. Workshop on Peer-to-Peer Systems*, number 2429 in LNCS, pages 22–33. Springer-Verlag, 2002.

[135] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor : A hunter of idle workstations. In *8th International Conference on Distributed Computing Systems*, pages 104–111. IEEE Computer Society Press, 1988.

[136] Julio López and David O'Hallaron. Support for interactive heavyweight services. Technical Report CMU-CS-01-104, Carnegie Mellon University, Feb. 2001.

[137] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *Procs 16th intl conf on Supercomputing*, pages 84–95. ACM Press, 2002.

[138] Qin Lv, Sylvia Ratnasamy, and Scott Shenker. Can Heterogeneity Make Gnutella Scalable ? In *1st Int. Workshop on Peer-to-Peer Systems*, number 2429 in LNCS, pages 94–103. Springer-Verlag, 2002.

[139] D. Narayanan M. Satyanarayanan. Multi-Fidelity Algorithms for Interactive Mobile Applications. *Wireless Networks*, 7(6) :601–607, 2001.

[140] J. MacLaren, R. Sakellariou, J. Garibaldi, , and D. Ouelhadj. Towards Service Level Agreement Based Scheduling on the Grid. In *Procs 2nd European Across Grids Conference*, 2004.

[141] C. Marchand, G. Da Costa, and O. Richard. Traces et profils utilisateurs dans l'ADSL, application aux systèmes Pair à Pair. http ://www-id.imag.fr/Laboratoire/Membres/Da-Costa_Georges/paper/ACI-GRID-CGP2P-2003.pdf, 2003. Présentation CGP2P.

[142] M.Beynon, R. Ferreira, T. M. Kurc, A. Sussman, and J. H. Saltz. DataCutter : Middleware for Filtering Very Large Scientific Datasets on Archival Storage Systems. In *IEEE Symp. on Mass Storage Systems*, pages 119–134, 2000.

[143] K. Nagaraja, R. Bianchini, R. Martin, and T. Nguyen. Using fault model enforcement to improve availability. In *Procs 2nd Workshop on Evaluating and Architecting System Dependability*, 2002.

[144] M. Neary and al. Javelin++ : Scalability Issues in Global Computing. In *ACM Java Grande'99 Conf.*, 99.

[145] M.O. Neary, S. P. Brydon, P. Kmiec, S. Rollins, and P. Cappello. Javelin++ : scalability issues in global computing. *Concurrency : Practice and Experience*, 12(8) :727–753, 2000.

[146] J. Nieh and M. Lam. The Design, Implementation and Evaluation of SMART : A Scheduler for Multimedia Applications. In *Procs 16 th ACM Symp. on Operating Systems Principles*, pages 184–197, 1997.

[147] D. Nurmi, J. Brevik, and R. Wolski. Modeling machine availability in enterprise and wide-area distributed computing environments. Technical Report CS2003-28, U.C. Santa Barbara Computer Science Department, 2003.

[148] A. Osorio, O. Traxer, S. Merran, J. Atif, and X. Ripoche. An augmented reality system for percutaneous nephrolithotomy. InfoRAD 2003 - RSNA'03, 2003. Award : Certificat of Merit.

[149] K. Park, G. Kim, and M. Crovella. On the relationship between file sizes, transport protocols, and self-similar network traffic. Tech. rep, Boston University, 1996.

[150] X. Pennec, P. Cachier, and N. Ayache. Tracking Brain Deformations in Time-Sequences of 3D US Images. *Pattern Recognition Letters - Special Issue on Ultrasonic Image Processing and Analysis*, 24(4-5) :801–813, 2003.

[151] T. Ping, G. Sodhy, C. Yong, F. Haron, and R. Buyya. A Market-Based Scheduler for JXTA-Based Peer-to-Peer Computing System. In *Procs Int. Conf. Computational Science and Its Applications (ICCSA 2004).*

[152] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Procs 9th annual ACM symp. on Parallel Algorithms and Architectures*, pages 311–320. ACM Press, 1997.

[153] J. Postel and J. Reynolds. Telnet protocol - rfc 854. http ://www.faqs.org/rfcs/rfc854.html, 1983.

[154] J. Postel and J. Reynolds. File transfer protocol - rfc 959. http ://www.faqs.org/rfcs/rfc959.html, 1985.

[155] J. Pruyne and M. Livny. Managing checkpoints for parallel programs. In *Workshop on Job Scheduling Strategies for Parallel Processing, IPPS'96.*

[156] J. Pruyne and M. Livny. Interfacing Condor and PVM to harness the cycles of workstation clusters. *Future Generation Computer Systems*, 12(1) :67–85, May 1996.

[157] S. Raman and S. McCanne. A model, analysis, and protocol framework for soft state-based communication. In *Procs Conf. on Applications, technologies, architectures, and protocols for computer communication*, pages 15–25. ACM Press, 1999.

[158] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Procs 2001 conf. on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172. ACM Press, 2001.

[159] J. Regehr and J. A. Stankovic. Augmented CPU Reservations : Towards Predictable Execution on General-Purpose Operating Systems. In *Procs. 7th Real-Time Technology and Applications Symposium (RTAS 2001)*, June 2001.

[160] O. Richard and F. Cappello. Sur la nature auto-similaire de l'activité de stations de trav ail et de serveurs HTTP. *Technique et science informatique*, 17(5) :635–658, 1998.

[161] Olivier Richard. *Contribution à l'étude des grappes de serveurs multiprocesseurs.* Phd, Université Paris-Sud, 1999.

[162] Matei Ripeanu, Adriana Iamnitchi, and Ian Foster. Mapping the Gnutella Network. *IEEE Internet Computing*, 6(1), 2002.

[163] A. L. Rosenberg. Guidelines for Data-Parallel Cycle-Stealing in Networks of Workstations, II ; on maximizing guaranteed output. *Foundations of Computer Science*, 11 :183–204, 2000.

[164] A. L. Rosenberg. Optimal Schedules for Cycle-Stealing in a Network of Workstations with a Bag-of-Tasks Workload. *IEEE Trans. Parallel Distrib. Syst.*, 13(2) :179–191, 2002.

[165] A. L. Rosenberg. Guidelines for Data-Parallel Cycle-Stealing in Networks of Workstations, I ; on maximizing expected output. *JPDC*, 59 :31–53, 99.

[166] L. F. G. Sarmenta. Sabotage-tolerance mechanisms for volunteer computing systems. *Future Generation Computer Systems*, 18(4) :561–572, 2002.

[167] L. F. G. Sarmenta, S. Hirano, and S. A. Ward. Towards Bayanihan : builing an extensible framework for Volonteer Computing using Java. In *ACM Workshop on Java for High-Performance Network Computing*, 1998.

[168] S.A. Savari and D.P. Bertsekas. Finite termination of asynchronous iterative algorithms. *Parallel Computing*, 22 :39–56, 1996.

[169] P. Schelkens, A. Munteanu, J. Barbarien, M. Galca, X. Giro i Nieto, and J. Cornelis. Wavelet Coding of Volumetric Medical Datasets. *IEEE Trans. Medical Imaging*, 22(3) :441–458, 2003.

[170] S. J. Sciutto. Aires : Air showers extended simulation. Department of Physics of the Universidad Nacional de La Plata, Argentina, 1995. http ://www.fisica.unlp.edu.ar/auger/aires/.

[171] A. Selikhov, G. Bosilca, C. Germain, G. Fedak, and F. Cappello. MPICH-CM : a Communication Library Design for a P2P MPI Implementation. In *9th Euro PVM/MPI*, pages 323–330, Oct 2002.

[172] S. Sen, J. Rexford, and D. F. Towsley. Proxy Prefix Caching for Multimedia Streams. In *3rd INFOCOM*, pages 1310–1319, 1999.

[173] V. Servois, A. Osorio, and J. Atif et al. A new pc based software for prostatic 3d segmentation and volume measurement. application to permanent prostate brachytherapy (ppb) evaluation using ct and mr images fusion. InfoRAD 2002 - RSNA'02, 2002.

[174] J. Shalf and E. W. Bethel. Cactus and Visapult : An Ultra-High Performance Grid-Distributed Visualization Architecture Using Connectionless Protocols. *IEEE Computer Graphics and Applications*, 23(2) :51–59, 2003.

[175] D. Siegmund. *Sequential Analysis*. Springer Series in Statistics. Springer Verlag, 85.

[176] S. Smallen, H. Casanova, and F. Berman. Applying scheduling and tuning to on-line parallel tomography. In *Procs 2001 ACM/IEEE conference on Supercomputing (CDROM)*, 2001.

[177] L. Smarr and C. E. Catlett. Metacomputing. *Communications of ACM*, 35(6) :45–52, June 1992.

[178] C. Smith. Open Source Metascheduling for Virtual Organizations with the Community Scheduler Framework (CSF). http ://www.platform.com/resources/whitepapers/, 2004.

[179] S. Son and M. Livny. Recovering Internet Symmetry in Distributed Computing. In *Procs 3rd Intl Symp. on Cluster Computing and the Grid*, 2003.

[180] A. Srinivasan and J. H. Anderson. Fair scheduling of dynamic task systems on multiprocessors. *Jal. of Systems and Software*, 03.

[181] A. Srinivasan and J. H. Anderson. Efficient scheduling of soft real-time applications on multiprocessors. *Jal. Embedded Computing*, 1(3) :1–14, 04.

[182] G. Stellner. Cocheck : Checkpointing and process migration for mpi. In *IPPS '96 : Procs of the 10th Int. Parallel Processing Symp.*, pages 526–531. IEEE Computer Society, 1996.

[183] A. Su, F. Berman, R. Wolski, and M. M. Strout. Using AppLeS to Schedule Simple SARA on the Computational Grid. *Int. Jal. High Performance Computing Applications*, 13 :253–262, 1999.

[184] V. S. Sunderam. PVM : A Framework for Parallel Distributed Computing. *Concurrency : Practice and Experience*, 2(4) :315–339, 1990.

[185] D. Thain, J. Bent, A. Arpaci-Dusseau, R. Arpaci-Dusseau, and M. Livny. Pipeline and Batch Sharing in Grid Workloads. In *Proc. 12th IEEE Symp. on High Performance Distributed Computing*, pages 152–161, June 2003.

[186] D. Thain, J. Bent, A. Arpaci-Dusseau, R.Arpaci-Dusseau, and M. Livny. Gathering at the Well : Creating Communities for Grid I/O. In *Procs of Supercomputing 2001*, Nov.

[187] Douglas Thain and Miron Livny. Multiple bypass : Interposition agents for distributed computing. *Journal of Cluster Computing*, 5 :39–47, 2001.

[188] Douglas Thain and Miron Livny. Parrot : An application environment for data-intensive computing. *Journal of Parallel and Distributed Computing Practices*, 2004.

[189] M. Unser, A. Aldroubi, and A. Laine (eds). Special Issue on Wavelets in Medical Imaging. *IEEE Trans. Medical Imaging*, 22(3), 2003.

[190] A. Uresin and M. Dubois. Asynchronous iterative algorithms : models and convergence. In *Advances in parallel algorithms*, pages 302–342. John Wiley and Sons, 1992.

[191] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms. *SIGOPS Oper. Syst. Rev.*, 36(SI) :239–254, 2002.

[192] T. Voinson, L. Guillemot, and JM. Moureaux. Image compression using lattice vector quantization with code book shape adapted thresholding. In *Proc. IEEE Int. Conf. on Image Processing*, Sept. 2002.

[193] A. Wald. *Sequential Analysis.* Wiley Series in Stat. Wiley, 66.

[194] Y. Wang, P. Chung, and W. Fuchs. Tight upper bound on useful distributed system checkpoints. Tech. Rep CRHC-95-16, University of Illinois at Urbana-Champaign, 1995.

[195] Y. Wang, P. Chung, I.J. Lin, and W. Fuchs. Checkpoint space reclamation for uncoordinated checkpointing in message-passing systems. *IEEE Trans. Parallel Dist. Systems*, 6(5) :546–554, 1995.

[196] Y.M. Wang. Consistent global checkpoints that contain a given set of local checkpoints. *IEEE Trans. Comput.*, 46(4) :456–468, 1997.

[197] Y.M. Wang and W.K. Fuchs. Lazy checkpoint coordination for bounding rollback propagation. In *IEEE Symp. Reliable Distributed Systems*, pages 78–85, 1993.

[198] Bryce Wilcox-O'Hearn. Experiences Deploying a Large-Scale Emergent Network. In *1st Int. Workshop on Peer-to-Peer Systems*, number 2429 in LNCS, pages 104–110. Springer-Verlag, 2002.

[199] W. Willinger, M. Taqqu, and A. Erramilli. *Stochastic Networks*, chapter A bibliographical guide to self-similar traffic and performance modeling for modern high-speed networks, pages 339–366. Oxford University Press, f. p. kelly, s. zachary and i. ziedins edition.

[200] R. Wolski, N.T. Spring, and J. Hayes. The Network Weather Service : A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Comp. Sys.* , 15(5-6) :757–768, 99.

[201] Z. Zhang, Y. Wang, D. Du, and D. Su. Video staging : a proxy-server-based approach to end-to-end video delivery over wide-area networks. *IEEE/ACM Trans. Networking*, 8(4) :429–442, 2000.

[202] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D Joseph, and J. D. Kubiatowicz. Tapestry : A Global-scale Overlay for Rapid Service Deployment. *IEEE Journal on Selected Areas in Communications*, 22(1) :41–53, 2004. Special Issue on Service Overlay Networks.

[203] Ben Y. Zhao, Yitao Duan, Ling Huang, Anthony D. Joseph, and John Kubiatowicz. Brocade : Landmark Routing on Overlay Networks. In *1st Int. Workshop on Peer-to-Peer Systems*, number 2429 in LNCS, pages 34–44. Springer-Verlag, 2002.

[204] Songnian Zhou. LSF : load sharing in large-scale heterogeneous distributed systems. In *Proceedings of the Workshop on Cluster Computing*, Tallahassee, FL, 1992. Supercomputing Computations Research Institute, Florida State University.