UNIVERSITÉ DU QUÉBEC

MEMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN MATHÉMATIQUE ET INFORMATIQUE APPLIQUÉES

PAR
GUY BOISCLAIR

EVALUATION DE LA PERFORMANCE DE CALCUL RÉPARTIS PAR FRAGMENTATION EN BANQUES DE CALCUL SUR DES SYSTÈMES NON-DÉDIÉS

SEPTEMBRE 2005

SOMMAIRE

Produire un modèle de gestion de répartition/répartiteur sur un réseau LAN avec synchronisation par processus redondants afin d'assurer l'optimisation de l'utilisation des ressources sur le calcul numérique du maillage automatique par la méthode des éléments finis.

Garantir une probabilité élevée de convergence. Application d'un modèle sectorisé en banques supervisées et intégrées en temps réels.

Evaluation de la performance de calcul répartis par fragmentation en banques de calcul sur des systèmes non dédiés

AVANT PROPOS et REMERCIEMENTS

Ce projet a pour objectif de mettre en place un sujet qui est actuellement en évolution et qui cherche encore un sens précis dans le monde qui nous entoure. Je remercie tous ceux qui m'ont aidés au long de ce tumultueux cheminement et qui m'ont permis de terminer cette œuvre, particulièrement ma famille, mon directeur et mes amis.

Ce travail est le début et la fin de ce qui est une erreur de parcours, non pas que le sujet n'est pas viable, ni que le travail d'implantation était trop complexe ou irréalisable, mais simplement qu'il semble que le contenant soit plus important que le contenu et c'est là quelque chose d'assez important pour ne pas continuer.

TABLE DES MATIÈRES

Page

Page

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

Page

INTRODUCTION

Nul n'aurait imaginé lors de la conception du premier calculateur balistique de l'impact qu'aurait l'électronique sur notre mode de vie. Le standard de signaux électriques qui porte le nom d'électronique numérique ou encore électronique digitale engendre une ère nouvelle au cours de laquelle l'ordinateur voit le jour. Au tout début, le premier « ordinateur » ne fait guère plus que de simples calculs et occupe l'emplacement d'une maison. Aujourd'hui, les mêmes fonctionnalités tiennent dans une montre à bas prix. L'ordinateur d'autrefois était dédié aux calculs militaires car son coût n'était pas à la portée des gens du grand public. L'évolution informatique pris son envol dans les années 60 dans le cadre du projet lunaire américain de la NASA et parallèlement du côté soviétique. La conception d'ordinateur eue à la fin des années 70, un nouvel essor avec l'apparition des premiers jeux et avec des consoles à prix assez abordable. Dans les années 80, l'ordinateur personnel apparaît dans certaines entreprises pour rapidement conquérir les domiciles des gens du domaine scientifique ou les gens plus fortunés. L'automatisation des procédés, les calculs qui hier étaient impensables et les jeux vidéos poussent l'ordinateur à ses limites et force un incroyable bond dans le domaine lors des années 90. Les coûts d'acquisition des appareils numériques, tels les ordinateurs, chutent, permettant une standardisation envers les gens de la rue. Ce qui hier était réservé aux calculs spécialisés et coûteux, est maintenant dans le domicile de plusieurs individus. Ce qui explique le remplacement des ordinateurs par les minis ordinateurs et le remplacement de ceux-ci par l'ordinateur personnel. Au début des années 90 plusieurs compagnies se spécialisèrent dans le calcul parallèle, c'est-à-dire des plates-formes dotées de plusieurs microprocesseurs permettant l'accélération des calculs numériques. Ces équipements pouvaient facilement coûter quelques millions de dollars à plusieurs centaines de millions de dollars et leur puissance de calcul n'étaient en rien comparable à ce qui s'offraient sur le marché « public ». Les réseaux de cette époque se résumaient à une simple liaison « MODEM » du côté public tandis que les réseaux étaient bien établis au niveau des institutions gouvernementales et scientifiques. Le concept de l' « INTERNET » fait

son apparition et celui-ci est adopté rapidement. Il crée une demande telle que l'industrie d'aujourd'hui cherche encore à la satisfaire. La demande en bande passante augmente du simple transfert de petits fichiers, à ce qui devient la parole numérique, l'imagerie etc. Les réseaux locaux apparaissent et ceux-ci permettent des débits de transfert de plusieurs millions de bits par seconde. L'industrie peut désormais se permettre l'acquisition des équipements réseaux en raison des coûts plus abordables. Ces derniers deviennent si abordables que tous peuvent maintenant se le permettrent. La propagation des réseaux et la diversité de leurs utilisations font en sorte que les débits de transfert passent de millions de bits/seconde à plusieurs milliards de bits/seconde. Les processeurs connaîssent une croissance semblable vers la fin des années 90, ce qui engendre un phénomène nouveau au niveau des amortissements en rentabilité. Jadis, l'amortissement informatique se faisait sur plusieurs années et maintenant il faut le faire sur une période n'excédant pas plus de trois ans. Alors qu'hier on pouvait se permettre l'acquisition justifiée d'équipements coûteux dans le but d'améliorer la performance, la règle change et il faut désormais opter pour une rotation rapide des équipements. Ce dernier phénomène engendre une diminution marquée de l'utilisation des appareils coûteux de type parallèle au profil d'un nouveau type d'architecture soit les architectures distribuées.

CHAPITRE 1

REVUE DE LITTÉRATURE

1.0 Définition : Qu'est-ce que l'architecture distribuée? [ATTIYA, 1998]

La base du principe est de prendre un calcul ou communément appelé, une charge de travail au sens informatique et de la scinder en plus petites sections de calcul afin de répartir ces sections sur différentes unités de calcul. De cette manière, nous prenons un processus dit séquentiel et nous le distribuons afin de le rendre parallèle dans le but d'améliorer le temps total de calcul. Cette dernière énumération se réfère cependant plus à la parallèlisation sur une machine à plusieurs processeurs.

Dans le cadre de l'architecture distribuée, il s'agît tout simplement de remplacer les multiples processeurs d'un même appareil par le processeur de plusieurs appareils. Il faut mettre en œuvre un système capable de tirer profit d'équipements reliés par un réseau. Les calculs sont fragmentés de manière semblable au modèle parallèle, mais envoyés vers différents appareils. De plus en plus, des parcs d'ordinateurs sont placés à la disposition d'une clientèle cible, par exemple dans les institutions reliées à l'éducation. Le potentiel de ces appareils n'est pas exploité, il y a de grandes périodes durant lesquelles la puissance de calcul n'est pas utilisée à son meilleur et il est même rare que le potentiel global approche de sa limite. Cette dernière observation n'est pas scientifiquement démontrée, mais elle est observable.

La notion de partage de la puissance des appareils en réseau est moins coûteuse que les appareils parallèles spécialisés. Ces derniers avaient jadis une espérance de vie de plusieurs années, mais la croissance du potentiel de calcul des processeurs fait en sorte que six mois suffisent à rendre obsolète les plus puissants appareils parallèles. Il n'est donc plus justifié de faire l'acquisition de ces coûteux équipements qui deviennent périmés dans la seconde. Il est maintenant temps de se pencher vers l'exploitation du potentiel de calcul des parcs informatiques mis en réseau, qui par le simple fait de remplacer un équipement, augmente la performance globale.

Leurs coûts sont faibles par rapport à leurs grands frères parallèles et rappelons-nous que leurs puissances ne sont guère plus en retard que de quelques mois. Ce qui nous permet d'obtenir une puissance globale plus imposante et moins coûteuse que l'acquisition d'appareils parallèles. Un système distribué peut être défini de la manière suivante :

« Un système distribué est un système dans lequel des appareils situés sur un réseau, communiquent et coordonnent leurs actions par passage de messages [Coulouris, Dollimore et Kindberg, 2001] »

Dans ce mémoire, nous combinons les architectures distribuées au maillage automatique par la méthode des éléments finis. Il s'agit d'un travail informatique appliqué à un calcul automatique de maillage dans le cadre d'une analyse de structure. Cette dernière est un parfait exemple de l'utilisation de l'informatique en Ingénierie (Mécanique dans ce cas).

1.1 La méthode des éléments finis

La méthode des éléments finis sert à résoudre des problèmes d'équations aux dérivées partielles dont les solutions exactes sont impossibles à déterminer. Une approche numérique au problème doit permettre de calculer une solution la plus exacte possible [François, 1998].

Le principe de cette méthode [Batoz et Dhatt, 1990] [Dhatt et Touzot, 1984] [Zienkiewicz, 1977] est de résoudre le problème d'équations aux dérivées partielles défini sur un milieu continu en définissant une formulation intégrale équivalente sur un milieu discrétisé. On appelle « maillage » l'espace discrétisé du domaine. Il est composé de noeuds. La solution est calculée uniquement en certains points (les nœuds) de l'espace grâce à des méthodes numériques. La solution générale en tout point est alors obtenue par interpolation de la solution du problème discret.

1.1.1 Estimations d'erreur

La méthode des éléments finis est une méthode numérique qui permet de trouver une solution approchée de la solution exacte. Il est important de pouvoir estimer l'erreur faite durant le calcul afin de valider ou d'invalider celui-ci. Les causes de ces erreurs sont multiples :

- erreur de discrétisation
- erreur de modélisation des conditions aux limites
- erreur de modélisation du phénomène étudié
- erreur d'intégration et d'interpolation numérique

1.1.2 Représentation des solides

Le travail est réalisé dans le cadre d'un projet de recherche ayant comme objectif de définir un modèle solide et d'éléments finis. Ce modèle unifié comporte toutes les données nécessaires à la réalisation du calcul.

Le but recherché dans cette expérimentation est de modéliser un solide qui peut avoir des équations représentatives complexes à l'aide d'un modèle de composition basé sur des formes dont les équations sont connues et dont la composition approche le solide d'origine. Plusieurs propositions de modèles ont vu le jour, nous survolons quelques-unes d'entre elles :

- Énumération exhaustive : Prendre un solide et le diviser en un ensemble de cubes identiques dont la taille fait varier la précision de l'approximation (figure 1.1).

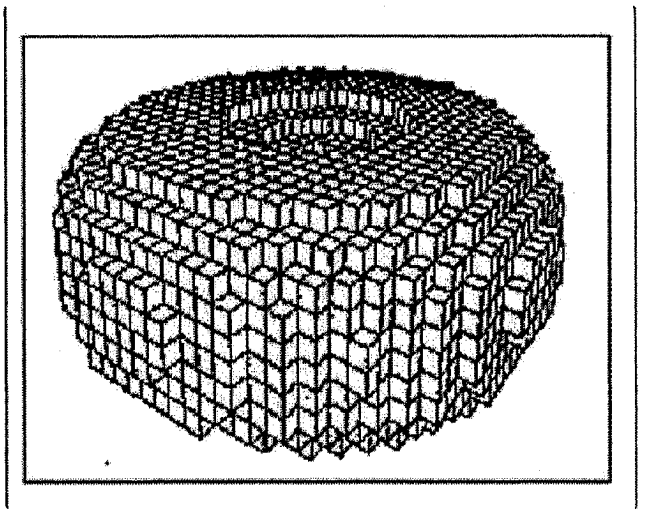

Figure 1.1 Énumération exhaustive (Mantyle 1988)

- Arbre octal : Ce modèle est un raffinement de la méthode précédente qui permet de varier la taille des cubes par secteur et ainsi améliorer l'approximation en minimisant l'information (Figure 1.2).

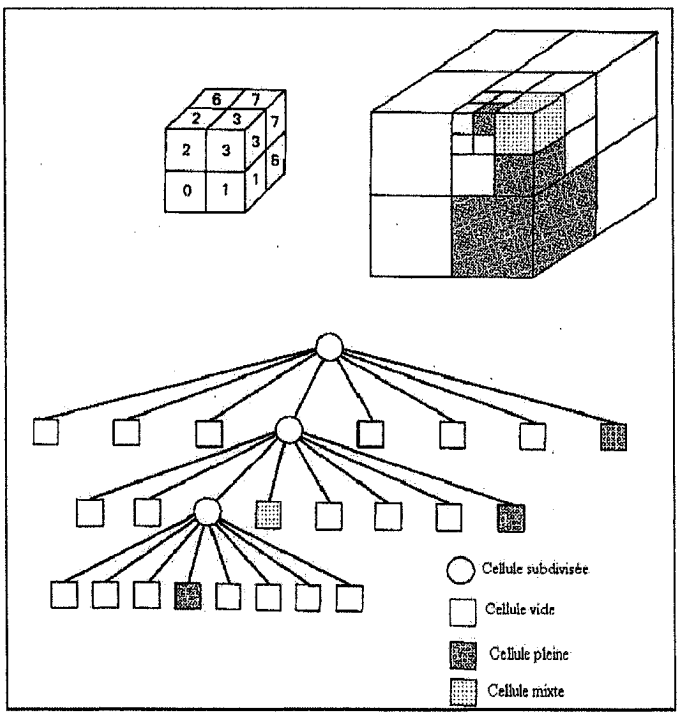

Figure 1.2 Le modèle octal. Modification de Mortenson

- Arbre CSG (Constructive Solid Geometry) : À partir d'un ensemble trivial de solides et d'opérations géométriques simples (union, soustraction et intersection), il s'agit de conserver l'historique d'utilisation des solides triviaux et des opérations exécutées sur ceux-ci afin d'obtenir le modèle du solide initial recherché (Figure 1.3).

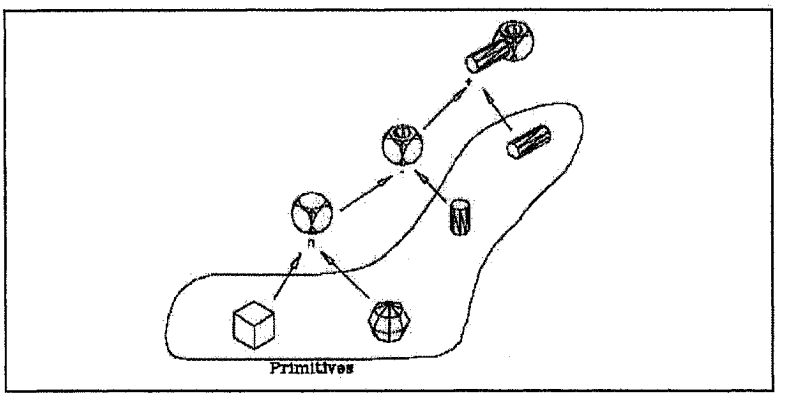

Figure 1.3 La modélisation par arbre CSG.

- Modélisation par les frontières (Boundary REPresentation) : Cette approche permet la modélisation du solide initial par une représentation des frontières dans R^3 (Figure 1.4).

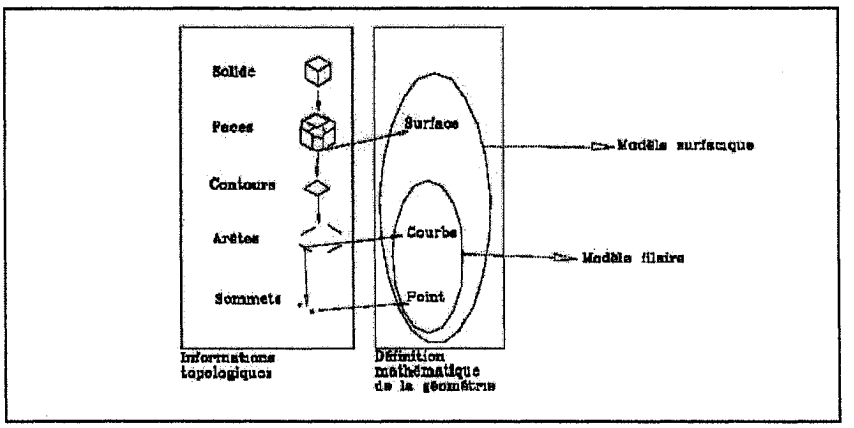

Figure 1.4 Description d'une structure par un modèle BREP.
On remarque que d'autres formalismes sont présents à l'intérieur de la structure BREP.

L'approche BREP permet la composition d'objets incluant des surfaces moins triviales telles que les surfaces de Béziers et les B-Splines (Figure 1.5).

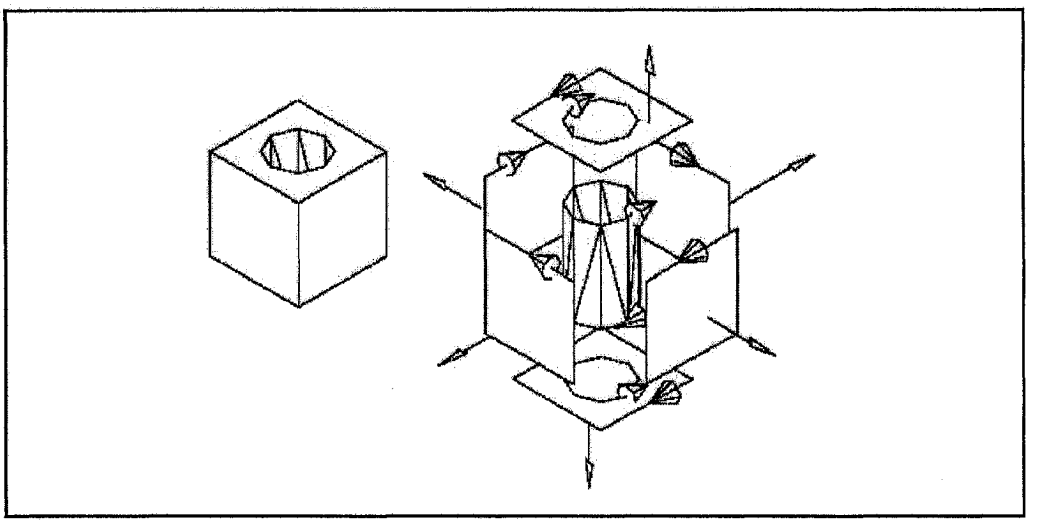

Figure 1.5 Orientation des surfaces à l'intérieur de la structure BREP.

- Modélisation Hybride : Elle est constituée par une composition découlant des approches CSG et BREP (Figure 1.6).

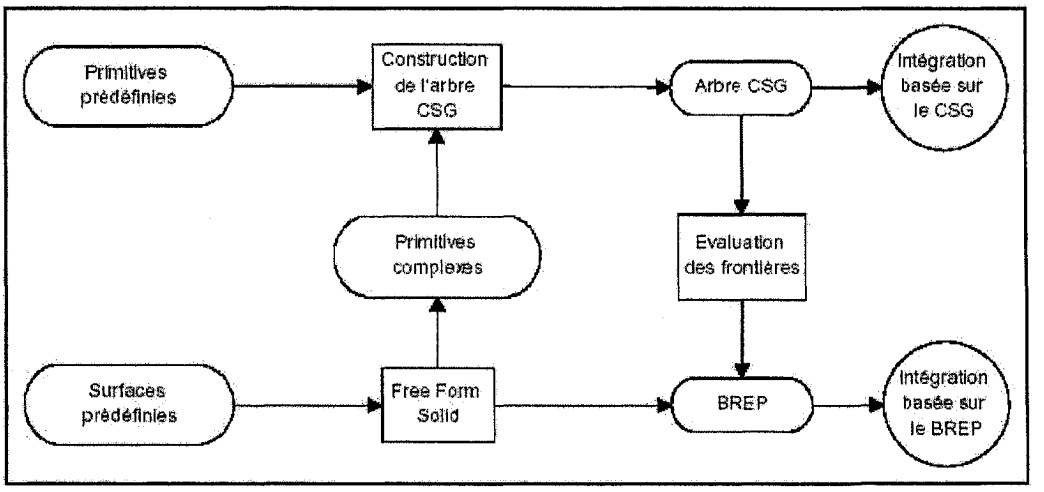

Figure 1.6 Le système de modélisation duale.

1.1.3 Techniques de maillage automatique :

Le maillage est la phase de division d'un modèle mathématique complexe en une suite de sous-modèles simples dont les solutions sont connues. Il est important de noter que cette division introduit une fois de plus, une approximation sur laquelle le « mailleur » doit travailler afin d'améliorer le résultat obtenu.

Observons trois propriétés du maillage (Figure 1.7):

- L'union des mailles doit reformer le modèle initial
- Deux mailles distinctes sont disjointes ou adjacentes
- Deux mailles adjacentes partagent intégralement leurs frontières.

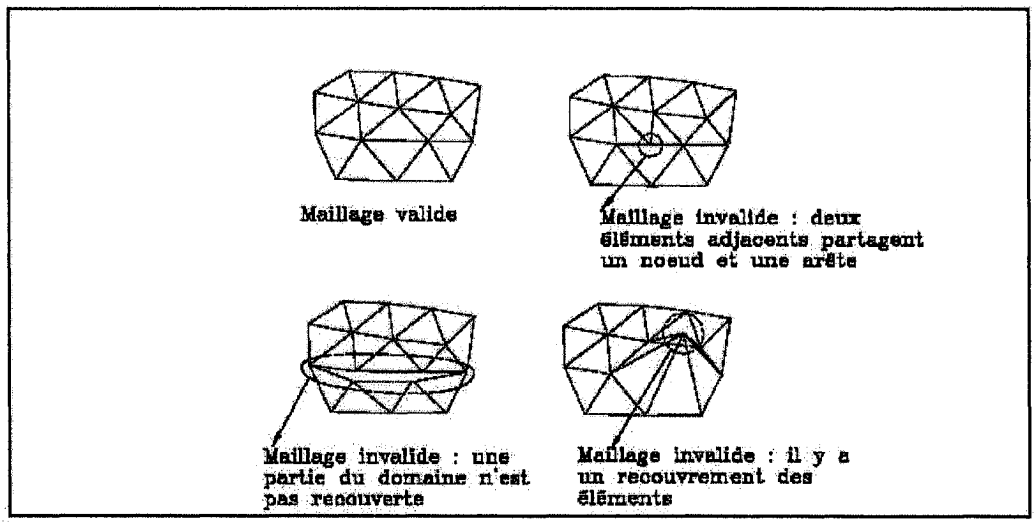

Figure 1.7 Les causes d'invalidité d'un maillage 2D.

La précision est obtenue sur la base de compromis entre le temps de calcul, les ressources utilisées et la finesse du maillage. Le triangle (tétraèdre 3D) est retenu comme élément de base pour sa facilité à être crée automatiquement.

1.1.3.1 Trois familles de méthodes de maillage :

Il existe deux aspects fondamentaux dans la production d'un maillage :

- La détermination de la position des nœuds : aspect géométrique
- La détermination de la connexion des nœuds : aspect topologique

Première famille : Elle repose sur la génération de la topologie et son adaptation à la géométrie du modèle. On y retrouve entre autre, la méthode de la décomposition spatiale.

Deuxième famille : Elle consiste à générer les nœuds avant de les connecter. Elle est appelée méthode de Delaunay-Voronoï et est basée sur l'insertion successive de nœuds en respectant le critère de Delaunay.

Troisième famille : La génération des nœuds et des mailles est simultanée. La méthode frontale entre dans cette catégorie.

1.2 Proposition d'un modèle de calcul distribué appliqué au maillage surfacique frontal pour la méthode des éléments finis

Dans le cadre de notre étude, nous étudions la performance temporelle des algorithmes de calcul du maillage surfacique frontal de solides sur un réseau d'ordinateurs non dédiés de type PC.

1.2.1 Maillage surfacique frontal

Le concept de maillage surfacique frontal est de calculer avec efficience pour chacune de ses faces, un ensemble de triangles (mailles) dont l'union reforme la surface et permet ainsi d'avoir accès à un calcul numérique sous forme de sommation d'objets simples pour le calcul par éléments finis.

Cette méthode n'est pas une méthode formelle mais plutôt intuitive, il s'agit d'élaborer à partir de la frontière de l'objet, les contours de la face vers son centre par itération successive jusqu'à ce que le domaine soit totalement couvert et ainsi triangulé. Un des problèmes se produit lorsque la frontière qui se déplace durant le procédé, entre en contact avec elle-même. Ce phénomène produit un effet de blocage et c'est ici que la résolution du blocage produit la portion intuitive du maillage et donne libre cours à la résolution de celui-ci. De fait, deux concepteurs ont probablement une approche de résolution des blocages différente l'une de l'autre et ceci produit une solution de fiabilité et acuité qui est en relation directe avec son concepteur.

1.3 Architecture distribuée à ce jour [SEETHARAMAN 1998] [BLAIR 1997] [REDMOND 1997].

L'apogée des architectures distribuées n'est pas encore passée, de nombreux ouvrages ont présentement cours, cependant ce domaine touche de près l'exploitation d'Internet, la révolution moderne de l'informatique. Le développement de la réseautique des dernières années dans le domaine de l'information et le commerce électronique a poussé plusieurs chercheurs et plus particulièrement l'industrie à mettre au point des solutions pour couvrir le besoin croissant de la répartition des tâches sur le réseau mondial. Plusieurs travaux du domaine commercial se conduisent sous le contrôle des multinationales qui ont pour une rare fois fusionnées sous le titre de comité de standardisation, tel « Object management Group OMG ». Ce consortium parmi d'autres vise à établir des standards dans le domaine croissant des architectures distribuées ou simplement le traitement des objets répartis. Généralement sous forme d'un « middleware » qui tient le rôle de couche d'abstraction de la répartition et du type des systèmes d'hébergement est disponible à des prix peu abordables. Le commerce électronique fait interagir une multitude d'appareils dont la classe matérielle et logicielle est rarement compatible.

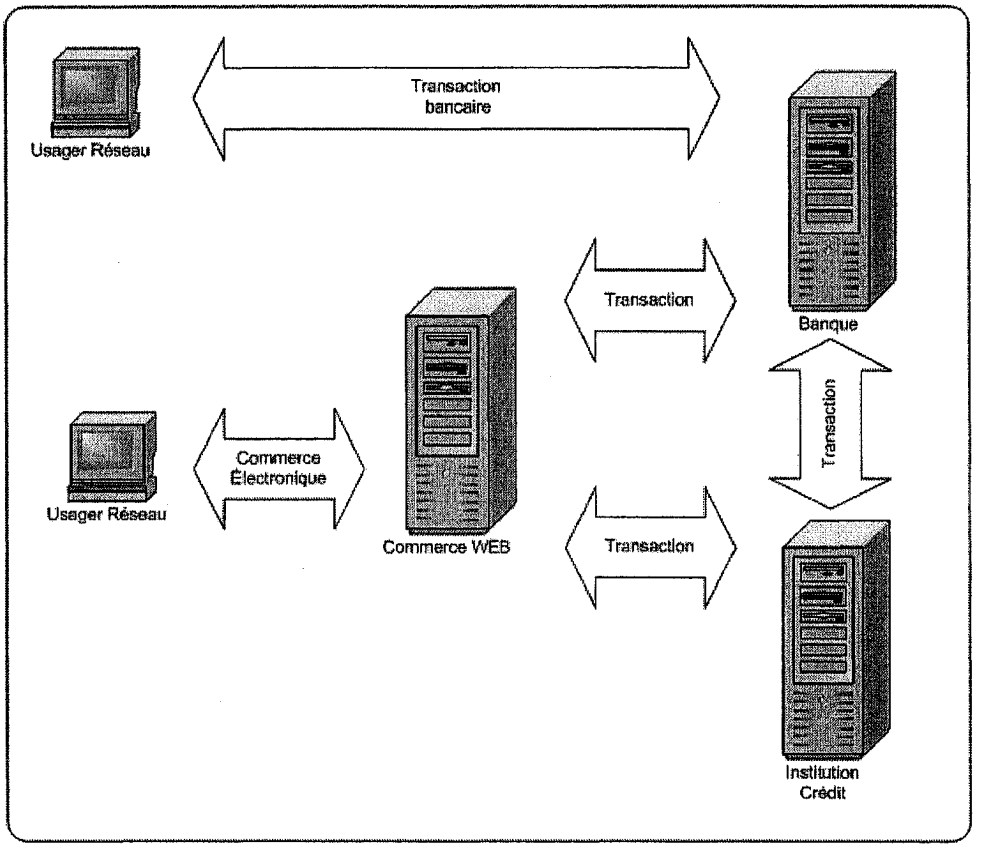

Figure 1.8 Modèle classique d'architecture répartie.

Les divers systèmes impliqués dans le modèle de la figure 1.8 démontrent un modèle dit de tiers parties qui implique l'interaction de plusieurs appareils dont l'hétérogénéité matérielle et logicielle dans la conduite d'une transaction « WEB » demande un degré d'abstraction par une passerelle logicielle sous forme de « bus ». Ce bus permet l'encapsulation des données dans le but de leurs transferts à travers différents équipements et ainsi compléter une transaction électronique bidirectionnelle.

Dans ce domaine, il existe plusieurs alternatives CORBA, Java RMI (...). Le sujet est en plein développement et des géants matériels et logiciels tels : IBM, Sun microsystem, Motorola, Intel, Microsoft et plusieurs autres, se réunissent sous forme de consortium [OMG] pour la mise au point combinée de normes dans ce domaine en pleine explosion.

1.3.1 La norme CORBA

L'exemple du Bus CORBA dans la figure 1.9 permet à des applications de diverses provenances en terme de langages de programmation et dont l'exécution se produit sur des systèmes hétérogènes tant au point de vue matériel que de leurs systèmes d'exploitation, de communiquer entre elles dans une grande translucidité.

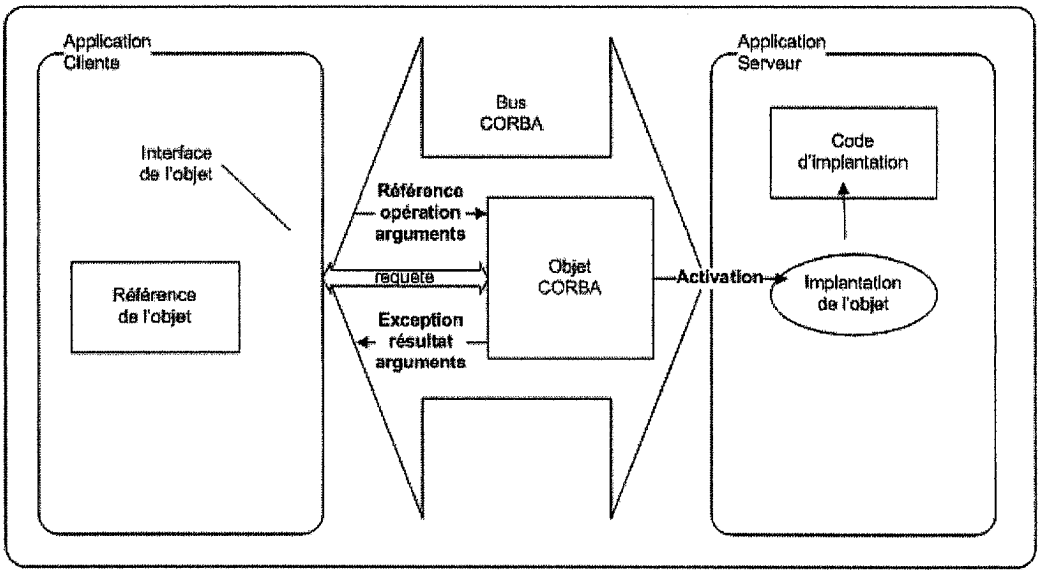

Figure 1.9 Le modèle objet de CORBA.

Ces solutions permettent dans l'architecture tiers une mise en œuvre de systèmes complexes et divers, cependant leurs coûts sont très élevés et les connaissances requises pour l'opération et le développement sous ces solutions demandent une grande connaissance du produit. De plus, une solution permettant une si grande translucidité des intervenants logiciels et matériels implique un passage d'abstraction élevée, ce qui n'est pas mal en soit, mais provoque des « constantes » de temps qui ne sont pas liées au temps de réponse des systèmes impliqués. Ces solutions s'adaptent au calcul distribué, mais une multitude de latences s'introduisent dans les temps d'opération et ainsi biaisent les résultats obtenus. De plus, ces solutions sont très lourdes en termes de ressources requises pour leurs exploitations et nécessitent des systèmes plus souvent qu'autrement dédiés.

1.3.2 Nommage [GUTTMAN,1999] [VAN STEEN,1998] [CHERITON, 1989]

Lors de transactions dans des systèmes distribués, la référence à un équipement distant se fait par le biais d'un alias (ex : URL). Un module de traduction sous forme de service (Figure 1.10) est donc requis pour l'identification physique de la destination qui ne peut être qu'une adresse IP car ce modèle est le plus répandu dans la réseautique locale et internationale (INTERNET). Pour la plupart des modèles actuels, une adresse fixe est souvent utilisée pour la source transactionnelle, bien que la technique de distribution de charge (load balancing) permette une répartition de la dite charge à travers un ensemble d'ordinateurs en augmentant la fiabilité et la disponibilité. Un tel système introduit via une couche d'abstraction logicielle, une abstraction de la transaction souhaitée. Ce module de traduction est un service de nommage et agit de manière abstraite et translucide afin de permettre aux applications une fluidité autant à la localisation qu'à la destination.

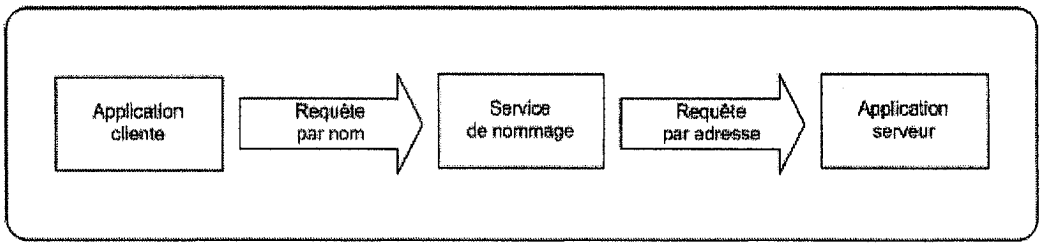

Figure 1.10 Modèle simple de nommage.

1.3.3 Java RMI [SUN]

La dernière décennie à procurée une explosion dans le domaine du modèle tiers et ainsi fait place à une demande au point de vue des langages de programmation d'application à interface WEB, ce qui donne un essor au niveau de Java qui est de par son origine un langage orienté objet de haut niveau très bien adapté à ce secteur d'activité. La partie « Remote Method Invocation » (RMI) du langage permet la définition de classes réseaux qui peuvent être appelées via cette interface. Cette distribution permet une souplesse de développement des applications WEB et de son ampleur, ce langage est disponible sur différentes plates-formes. Java RMI utilise la

sérialisation de l'information afin de permettre des échanges entre systèmes hétérogènes (Figure 1.11).

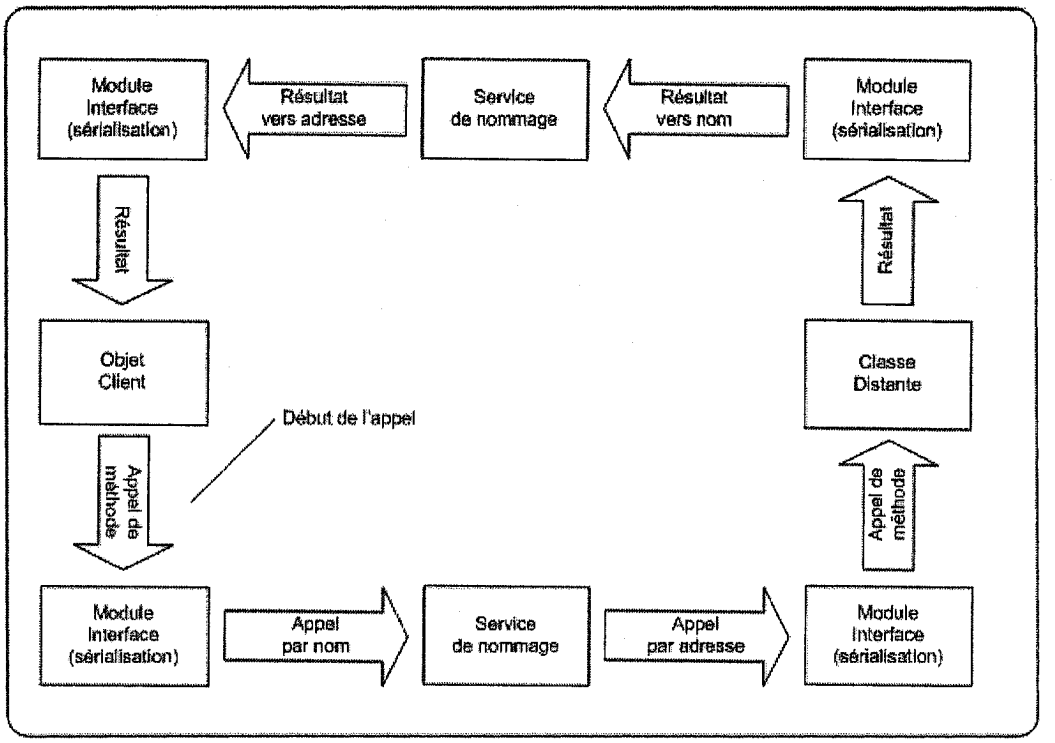

Figure 1.11 Le concept de JAVA RMI (Cycle d'appel).

La sérialisation n'est pas un concept unique à JAVA, elle est utilisée dans tous les modèles actuels de communication dans les architectures réparties, nous explorons ce concept à la section 1.3.9.

1.3.4 Réplication [WEISMAN, 2000] [STEELING, 1998] [EL ABBADI, 1985]

La réplication de données dans un système informatique permet d'augmenter la robustesse des systèmes en augmentant la tolérance aux pannes par duplication de l'information (Figure 1.12). Cette méthode est largement répandue dans les systèmes où l'information est critique, il n'y a qu'à penser à un système bancaire qui ne peut se permettre une perte d'information. Chaque transaction effectuée produit un effet de duplication dont la multitude est reliée à l'évaluation de la criticité de l'information. Il existe plusieurs méthodes de réplication, passant du modèle en temps réel, au modèle différé par queues de transactions.

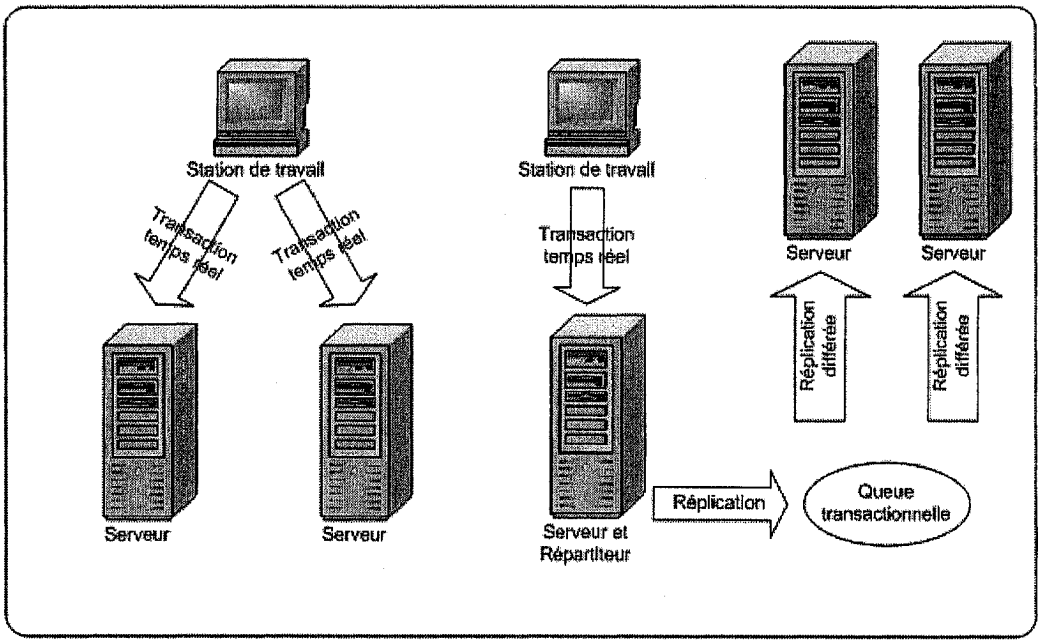

Figure 1.12 Réplication en temps réel et différée.

1.3.5 Latence [SHMIDT, 1998] [KONSTANTAS, 1997] [FIDGE, 1991]

La latence dans les systèmes informatiques est souvent comparée à une constante d'initiation d'un procédé, c'est le temps de mise en marche d'une opération sans pour autant être partie intégrale de l'opération (Figure 1.13). Ce temps est cependant variable mais d'un domaine de temps limité, c'est pourquoi il est souvent référé en tant que constante. Plus précisément dans un modèle parallèle ou d'architecture répartie, la latence avant le calcul est le temps de décomposition d'une opération en ses composants parallèles et la latence après calcul est la reconstruction des solutions partielles parallèles en solution totale. Ce temps inclut aussi dans notre étude, le temps de transfert des données, l'initiation du ou des protocoles de communication et de transaction, la distribution et la gestion de cette distribution et la gestion des disponibilités des équipements de calcul.

Un phénomène à ne pas négliger dans le concept de la latence est la synchronisation des évènements, puisque les opérations sont pour la plupart asynchrones, une base de référence doit être utilisée afin de s'assurer de la validité des résultats.

Il faut donc avoir une base commune de référence sous forme d'horloge synchronisée inter –appareils, une synchronisation des horloges de tous les postes du modèle réparti doit se faire et chaque transaction effectuée sur le système distribué doit porter le sceau normalisé sous forme de timbre temps universel à notre système de calcul.

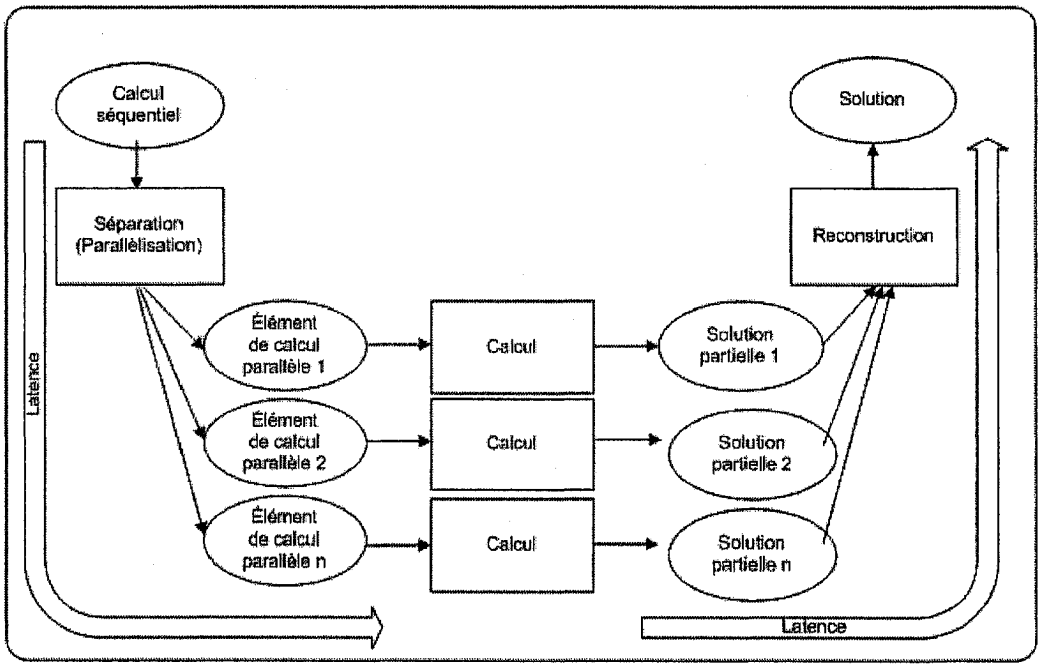

Figure 1.13 Latence dans la séparation et reconstruction d'un calcul parallèle.

1.3.6 Protocole transactionnel [WEIKUM, 1992] [EL ABBADI, 1985] [MOSS, 1985]

Dans les systèmes distribués de tierce partie, les transactions sont effectuées sous forme séquentielle, chaque service évaluant son champs d'action et donnant une suite récursive vers des services d'autre tiers. Il y a cependant une autre méthode qui consiste à établir un protocole supérieur qui se servira d'un porteur, tel TCP ou UDP pour la communication et d'unités transactionnelles qui servent à l'encryptage et le décryptage de ce protocole de plus haut niveau. Cela consiste à définir un protocole propre à l'application visée afin de diminuer la gestion de communication et établir ainsi une couche d'abstraction sur le réseau et ses méthodes (Figure 1.14).

Il est possible d'inclure plusieurs unités de traitement de ce protocole transactionnel à différents paliers et ainsi devenir une base pour une architecture distribuée avec possiblement un service intégré de résolution de noms. Ce protocole transactionnel vient s'insérer à l'appel de fonctions ou méthodes dans le cas d'un système orienté objets et il est la base même des modèles tels que CORBA de l'OMG, JAVA RMI de Sun ou Visual studio .NET de Microsoft. Ce protocole comprend une suite de macro-instructions qui lui sont propres et de méthodes protocolaires afin d'assurer son utilisation tout en étant abstraite à l'utilisateur.

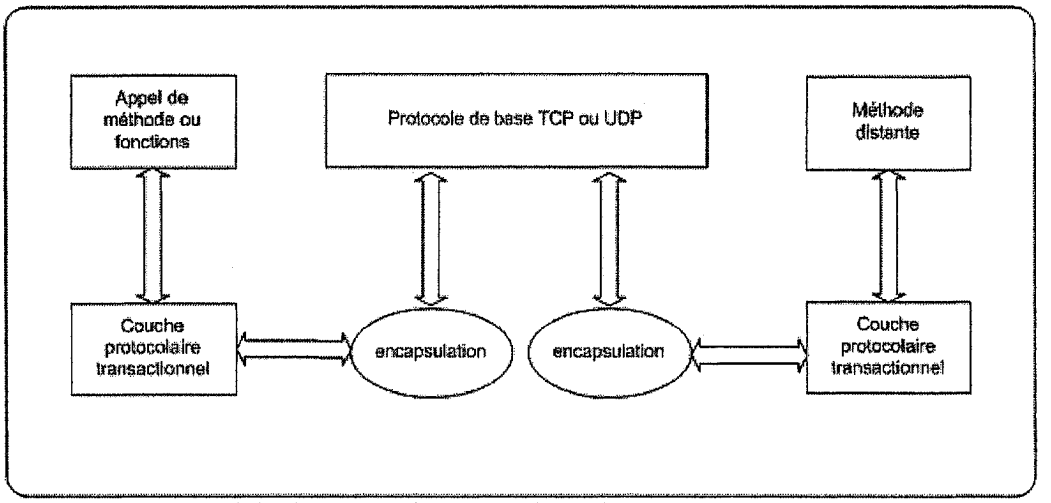

Figure 1.14 Modèle de protocole transactionnel.

1.3.7 Congestion réseau [NAGEL 1984]

Les réseaux modernes se basent pour la plupart sur une couche réseau local de protocole Ethernet, c'est à dire un protocole à gestion de collisions. Des collisions se produisent sur le réseau lorsque deux sources tentent une transmission simultané et l'absence de réponse du destinataire de notre message signifie la non réception du message et la conclusion d'une collision. Une retransmission est nécessaire et peut à nouveau causer une collision.

Ceci implique une quiétude partielle dans des réseaux de faible importance, mais qui introduit une problématique de taille dans les segments comptant plusieurs dizaines d'appareils. Pour remédier à cette situation, plusieurs méthodes existent, nous en couvrons sommairement quelques-unes.

L'élimination des protocoles bavards tels : NetBeui et Appletalk (dans sa version Ethertalk), permet de diminuer le nombre de collisions qui surviennent avec l'utilisation d'Ethernet. Cette solution est de niveau administratif et permet des gains appréciables au niveau de la décongestion réseau.

Il y a d'autres alternatives, cette fois matérielles telle la partition réseau par des passerelles qui limitent les échanges de types locaux, basés sur une connaissance des échanges inter appareils. Il est ainsi possible d'améliorer les performances, cependant une connaissance plus approfondie de notre réseau doit être maîtrisée et il est parfois impossible d'identifier les échanges dans le cas d'utilisation de tiers de niveau d'abstraction élevée. La translucidité apportée par ces tiers nous éloigne de la connaissance du cheminement réel. Ces méthodes ont les meilleurs rendements en dépit de l'ignorance des échanges car plusieurs diffusions sont interceptées dans les segments locaux de nos réseaux.

Il existe encore une autre méthode qui n'est pas étrangère aux deux précédentes et qui consiste à limiter l'utilisation des multi diffusions (broadcast). De ce côté, il faut maîtriser de manière radicale, l'utilisation faite par les différents composants réseaux ainsi que celle de leurs dérivés.

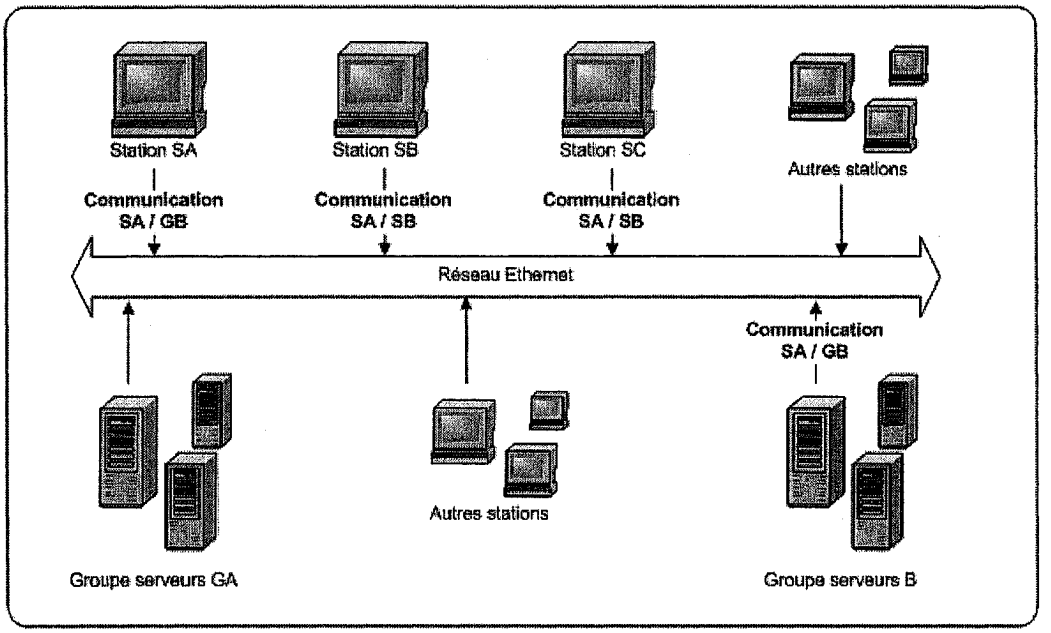

Figure 1.15 Congestion réseau.

Dans l'exemple de la figure 1.15, le simple fait que lors de la communication entre SA/SB, SB répond à SA lors d'une communication comportant plusieurs messages de SA peut engendrer une collision entre ces deux appareils sur le réseau.

1.3.8 Répartiteurs et serveurs de rôles [SANDHU 1996] [BERSHAD 1993]

Lorsque nous parlons de bus dans les architectures réparties, le concept de répartiteur (Broker) apparaît rapidement. Généralement, le répartiteur agît en tant que service de nommage ou au mieux en tant que base de connaissance pour la redirection des méthodes invoquées par les applications clients au bus. Le répartiteur doit avoir une connaissance des ressources disponibles sur le bus afin d'assurer la bonne conduite lors de l'invocation de méthodes distantes ou simplement la référence à des objets distants. Cette information est tenue à jour par l'identification ou l'enregistrement des disponibilités de la part des différents services constituant le bus (Figure 1.16). Cette disponibilité se traduit directement par une inscription des ressources offertes dans une base de données commune aux intervenants sur le bus tant aux niveaux, services que client.

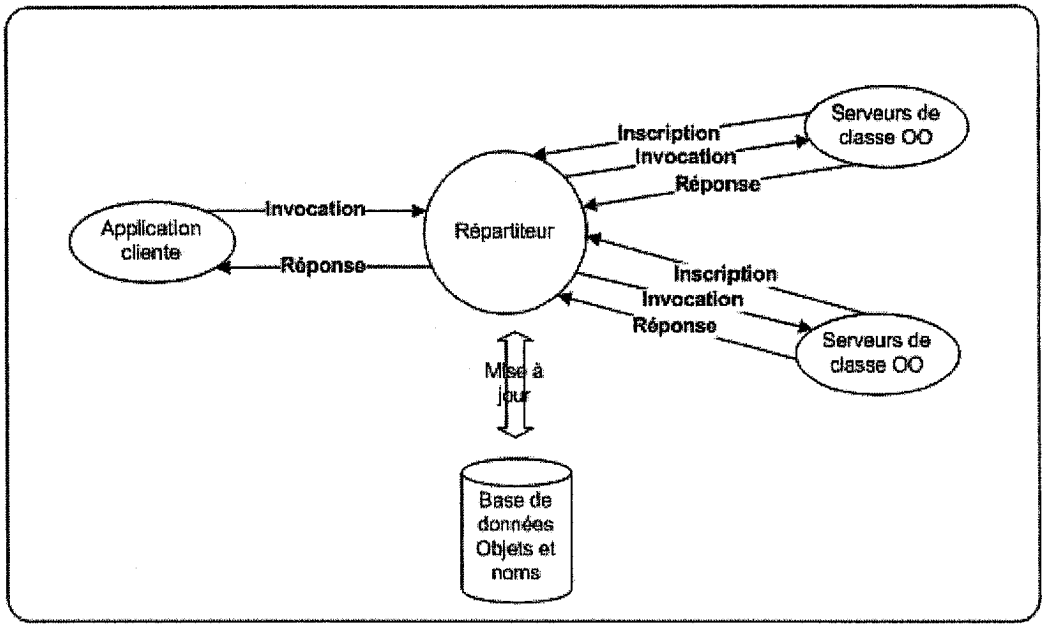

Figure 1.16 Inscription et utilisation des ressources.

Dans les architectures à plusieurs tiers, il est souvent probable que des entités offrant un service sur le bus, fassent appel à d'autres services pour la conduite des transactions qui leurs sont demandées, comme la résolution de noms. Dans ce cas, cette base d'information se doit d'être constamment à jour afin d'assurer la quiétude gestionnelle parmi le bus. Les répartiteurs agissent comme des méga services qui assurent le bon fonctionnement de haut niveau vers les services impliqués dans une transaction réseau.

Il existe un autre mode de répartition, c'est la répartition de charge dans les systèmes qui sont souvent sollicités. En effet, plusieurs services réseaux sont offerts au grand public, il n'y a qu'à penser à un site WEB transactionnel de grande multinationale, il est visité plusieurs milliers de fois par minute, cette charge écrase le plus performant des systèmes actuels. C'est pourquoi, des grappes d'appareils (cluster) sont formées pour répondre à ces grandes demandes (Figure 1.17), le concept est utilisé dans plusieurs domaines depuis un certain temps, ne serait-ce que pour obtenir quorum sur une décision ou simplement pour séparer une charge trop lourde pour un seul appareil.

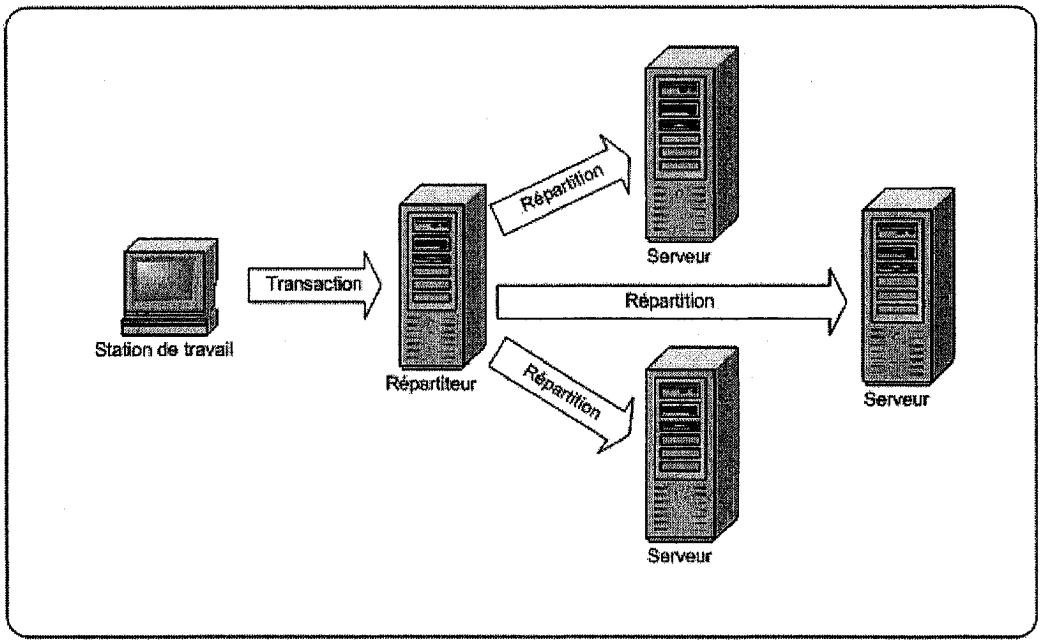

Figure 1.17 Modèle de répartition de charge.

1.3.9 Sérialisation

La sérialisation est un concept assez nouveau dans le domaine de l'informatique de réseau afin d'assurer la communication entre systèmes qui peuvent ne pas être compatibles au niveau de la représentation des objets de base. La représentation des entiers sous architecture matérielle Motorola et Intel est un exemple, le concept du « Little endian / Big Endian » qui permute les octets des valeurs entreposées en mémoire des entiers ou bien encore de la représentation « Unicode / ANSI » des chaînes de caractères sont d'autres exemples. Pour remédier à cette situation, il faut donc traduire toutes les demandes entre constituantes du réseau tant au niveau client que serveur en se rappelant que certains services font appel à d'autres. Nous passons donc du mode natif d'une application en mode universel via le biais d'une couche d'abstraction qui est la sérialisation (Figure 1.18). Toutes les invocations sur le bus sont faites dans ce mode universel qui facilite la communication, cependant cette traduction introduit une charge en temps à notre constante de latence.

Rappelons-nous cependant que ce modèle universel n'est vrai que dans le domaine de notre BUS, à moins d'utiliser des protocoles comme SOAP ou XML qui ne font pas partie de notre étude.

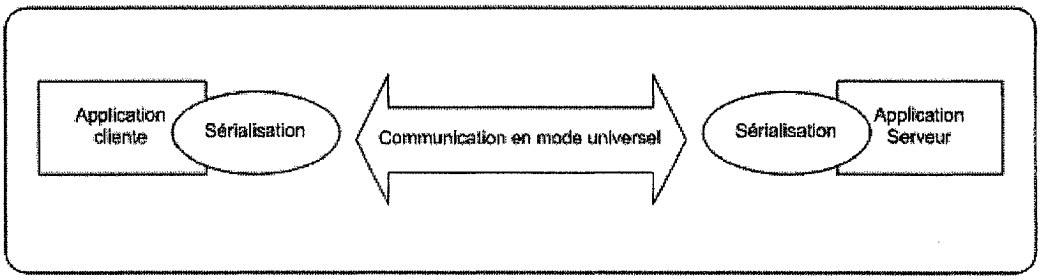

Figure 1.18 Abstraction par sérialisation.

1.3.10 Nomadicité [KLEINROCKS 1997]

Ce terme dérive essentiellement de la traduction latine entre sédentaire et nomade : demeurer à un endroit fixe ou être perpétuellement en mouvement avec de brefs arrêts. Dans le cas des appareils informatiques sur un réseau, il existe une relation directe avec le type d'équipement, par exemple, un portable est considéré comme nomade alors qu'un appareil de bureau est considéré comme sédentaire. Cependant dans un modèle d'architecture distribuée pour le calcul, la disponibilité de l'appareil, rend un sédentaire, nomade, par sa seule mise hors tension. Nous étudions la disponibilité de nos équipements sous deux aspects qui ont le même effet sur notre modèle, soit l'absence par mise hors tension ou encore par l'utilisation de celui-ci à des fins personnelles. Les appareils visés par notre étude ne sont pas dédiés au calcul, mais sont rendus disponibles au calcul en phase de non-utilisation par son utilisateur. Nous voyons plus loin les effets de cette indisponibilité tant au début des calculs que durant ceux-ci.

1.4 Problématiques des architectures distribuées

1.4.1 Temps d'accès

Un des problèmes majeurs des architectures distribuées dans les grands calculs vient du transfert d'information qui se fait à des échelles de vitesse largement inférieure à la rapidité de calcul des processeurs. Sans connaître toutes les subtilités de la conception des ordinateurs, il est facilement notable que pour une base de temps fixe et à un moment précis, le facteur entre la vitesse du processeur et celle du réseau variera entre 100 et 1,000. Ce même facteur passe au alentour de 100 pour les auxiliaires mémoires RAM. Nous débutons donc une petite étude arithmétique simple afin de bien comprendre le phénomène et ainsi avoir une idée des aspect à considérer dans l'étude. Nous prenons pour hypothèse le facteur de rapidité de 100 entre l'accès à la mémoire vive et le transfert réseau ce qui est tout à fait près de la réalité.

Hypothèse de travail :

- Le temps d'accès à la mémoire est fixé à 100,000,000 d'octets par seconde.
- Le temps de transfert réseau est de 1,000,000 d'octets par seconde.
- Les processeurs de tous les appareils sont de mêmes calibres.

Dans un calcul prenant 100 secs à s'exécuter séquentiellement et demandant un transfert de 100,000,000 d'octets de données. Comparons-le aux cas d'utilisations réseaux suivants :

1.4.1.1 Un seul appareil réseau de calcul

Le temps de transfert de 100,000,000 d'octets à raison de 1,000,000 octets/seconde est de 100 secondes. En considérant que le calcul doit être exécuté sur un seul appareil et que le transfert des données vers celui-ci doit être réalisé, les temps sont alors illustrés dans la figure 1.19.

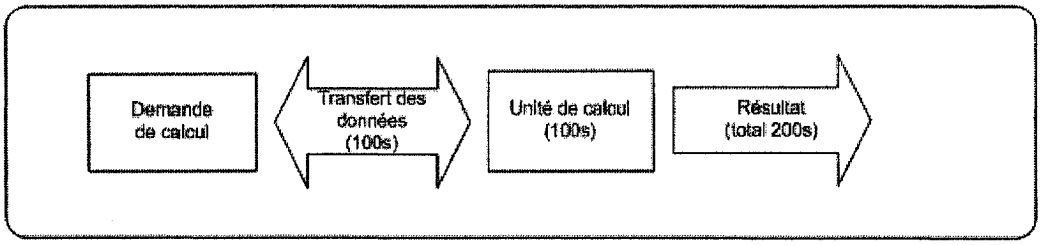

Figure 1.19 Cas d'utilisation, un seul appareil de calcul.

Une perte de performance est engendrée par rapport au modèle séquentiel.

1.4.1.2 Dix appareils, calculs indépendants

Nous disposons de dix appareils et nous pouvons fragmenter le calcul en dix segments de calcul égaux et indépendants au point de vue calcul. Le phénomène réagit avec les temps illustrés dans la figure 1.20.

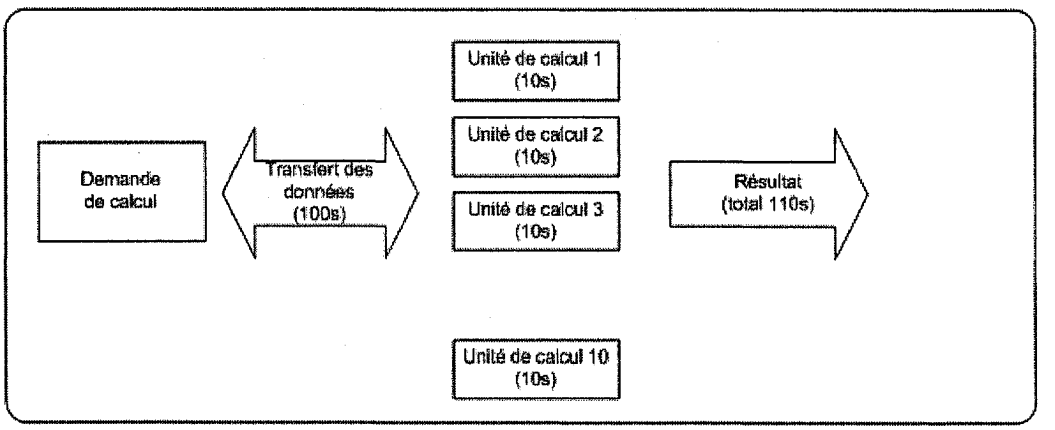

Figure 1.20 Cas d'utilisation, dix appareils, calculs indépendants.

Une perte de performance est toujours engendrée par le transfert des données, mais le calcul est globalement plus efficace comparativement au modèle séquentiel.

1.4.1.3 Dix appareils, calculs et données indépendants

Nous disposons de 10 appareils et nous pouvons fragmenter le calcul en dix segments de calcul égaux qui sont indépendant au point de vue des données. De plus, il est possible de sérialiser l'envoi des données de manière que le transfert des données pour le second calcul se fait durant le premier calcul.

Ici, nous voyons la possibilité de tendre vers une solution plus viable que la solution séquentielle bien que le résultat soit le même (Figure 1.21).

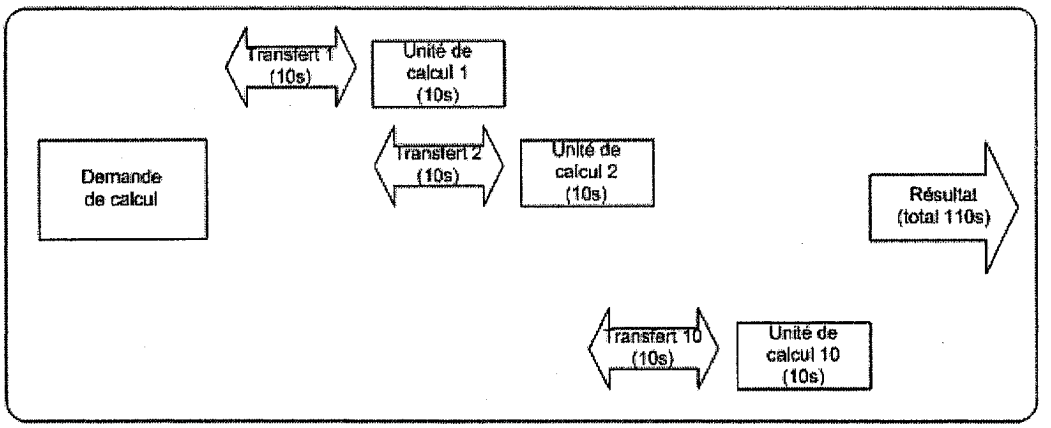

Figure 1.21 Cas d'utilisation, dix appareils, calculs et données indépendants.

Une perte de performance est toujours engendrée, mais malgré tout un potentiel se dégage de ce cas d'utilisation.

1.4.1.4 Dix appareils, dix mêmes calculs indépendants de données identiques

Dans bien des cas, il s'agit de lancer une itération de calcul sur un ensemble de données afin de trouver une solution optimale. Alors, les données sont les mêmes pour tous les calculs et les calculs sont semblables, c'est-à-dire, un même calcul avec des paramètres particuliers d'exécution. Le calcul séquentiel sur une itération de 10 calculs est C1(100s) + C2(100S) + … + C10(100s) = 1000s

Le résultat en temps de calcul sur un modèle distribué est illustré à la figure 1.22.

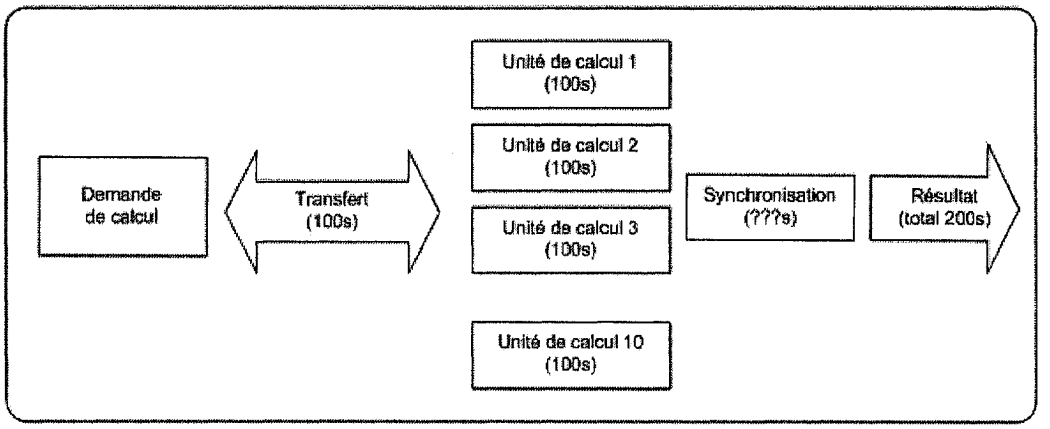

Figure 1.22 Cas d'utilisation, dix appareils, 10 mêmes calculs indépendants de données identiques.

Cette fois nous apercevons un gain appréciable, mais une notion de synchronisation fait son entrée avec un temps encore indéterminé mais qui est probablement de l'ordre de quelques secondes voire au plus une vingtaine. Notre étude tente à l'aide du modèle de maillage surfacique frontal d'établir les variations de performance et d'obtenir un gain de performance.

1.4.2 Parallélisation automatique

Nous remarquons que l'histoire du calcul parallèle a démontré que cette solution n'est applicable que dans un nombre limité de situations et que toutes les tentatives de parallélisation automatique des modèles de calcul n'ont donné qu'une faible réussite. Ce n'est pas difficile de comprendre la situation, pour paralléliser un calcul, il faut connaître les inter-dépendances des données et ce phénomène est très difficile voir impossible (du moins au moment présent) à automatiser. L'humain prouve encore une fois sa supériorité par rapport à la machine de manière très ponctuelle dans le temps. Il est très difficile à quelqu'un connaissant le modèle de calcul de trouver les inter-dépendances des données et cette tâche peut prendre un temps considérable.

Alors le but de cet ouvrage n'est pas de fragmenter le calcul, mais plutôt de prendre quelques modèles mathématiques déjà fragmentés et de lancer ces calculs sur une architecture distribuée en réseau.

1.5 Modèle proposé : général

Nous démontrons de manière structurée que les architectures distribuées peuvent apporter dans des cas précis, un gain en performance, mais que ce gain ne s'applique qu'à des types de calculs bien particuliers. Nous allons faire une analyse comparative des résultats obtenus en respect d'une proportion entre le temps de calcul et les données nécessaires au calcul, en illustrant le phénomène de cette dite proportion. Plusieurs utilitaires existent sur le marché, tant au niveau commercial que scientifique, qui fournissent des résultats qui dans la plupart des cas doivent être obtenus sur des ensembles dédiés d'équipements réseaux et d'unités de calculs. Nous tentons cet exercice sur un ensemble non-dédié d'appareils en utilisant un modèle adaptatif tant au point de vue réseau que de la répartition du calcul. Chaque unité de calcul peut à tout moment être retirée du circuit de disponibilité et il faut donc palier à cette situation en prévoyant des duplications de calculs en les répartissant sur plusieurs unités, espérant obtenir une complétude de calcul basée sur un modèle probabiliste trivial (Figure 1.23). Cette probabilité est obtenue en produisant un sondage d'utilisation sur les appareils pour une période donnée. Nous travaillons sur plusieurs modèles réseaux afin d'évaluer les variances de performances en fonction des différents équipements réseaux disponibles en proposant des estimés relatifs à la popularité et disponibilité de ceux-ci. Les transferts de données sont le facteur prépondérant de succès de notre modèle, c'est pourquoi, une attention particulière est portée à ce phénomène et l'étude des mémoires partagées en réseau est partie intégrale de notre étude. La plupart des produits existants pour la mise en œuvre de répartitions réseaux sont lourds et demande des installations et des équipements particuliers, de plus cette lourdeur à une incidence assez élevée sur la performance obtenue et nous devons dans le cadre de notre étude qui est basée sur la performance, éviter de biaiser nos résultats par des constantes de temps liées au produit.

Nous produisons donc un modèle sobre pour notre architecture distribuée afin de rendre les résultats de calcul le moins biaisés possible.

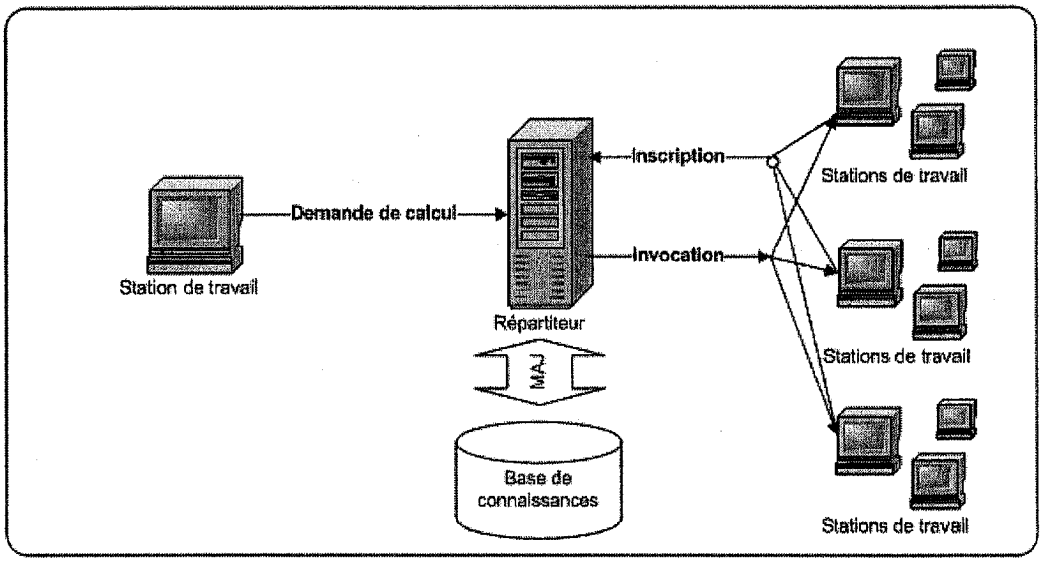

Figure 1.23 Le répartiteur, inscription et invocation.

1.5.1 Modèle proposé : unité de calcul

Nos premiers essais se font sur des unités dédiées de calcul dans un réseau fermé afin de produire un étalon pour notre modèle de base d'architecture distribuée. Cependant, le but ultime est de mettre ce modèle sur un ensemble d'appareils qui ne peuvent pas être considérés comme entièrement disponibles. Nous faisons notre étude sur des appareils servant dans des salles publiques, pouvant donc être utilisés à tout moment. En conséquence, il ne faut jamais limiter l'utilisation aux individus (Figure 1.24). La priorité d'exécution pour ces postes de travail est donnée aux utilisateurs des salles publiques et non au calcul distribué. Ceci implique qu'à un moment précis dans le temps, l'appareil est considéré disponible uniquement si personne n'y travaille. Cette disponibilité est temporelle car quelqu'un peut venir y travailler à tout moment.

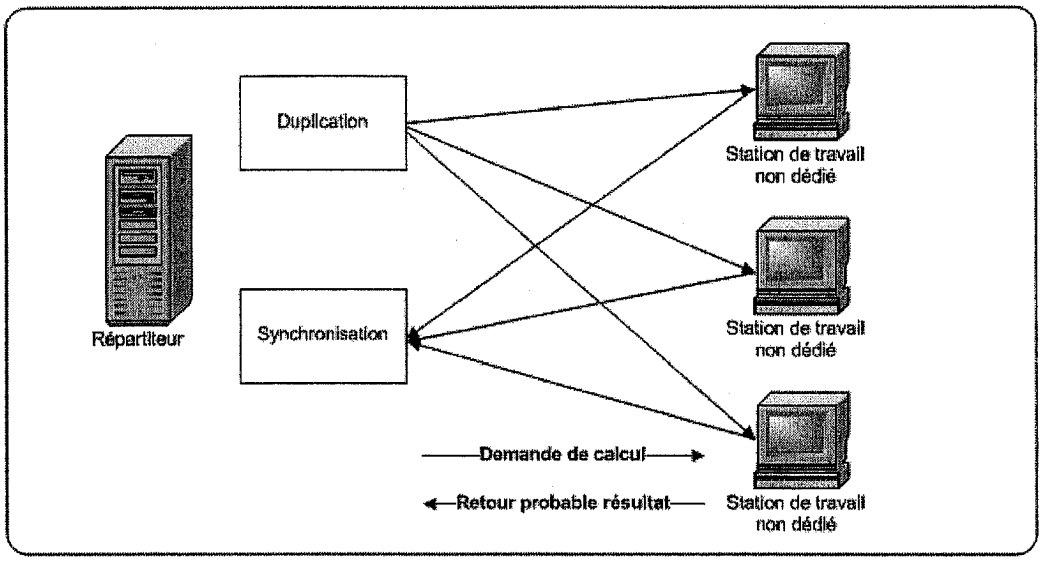

Figure 1.24 Le répartiteur, duplication des calculs.

C'est pourquoi un calcul peut avoir débuté et ne jamais se terminer en fonction d'une utilisation potentielle de l'appareil par un utilisateur à tout moment. Le parc visé compte divers types d'équipements en terme de capacité et performance, c'est pourquoi une base de connaissances sert de référence pour l'identification du potentiel de répartition afin d'en produire une estimation avant calcul de la distribution. Dans ce mode de fonctionnement, il est évident qu'en raison de la prise de contrôle par un usager potentiel à tout moment, le même calcul doit être lancé sur plusieurs unités afin d'assurer une probabilité élevée de terminaison d'un calcul donné. Cette approche demande une synchronisation du produit de calcul afin de ne pas dupliquer les résultats dans le cas où plusieurs unités viendraient à compléter leurs calculs. La base de connaissances doit être dynamique, permettant l'ajout et le retrait d'unités (ordinateurs) pour la disponibilité de calcul.

1.5.2 Modèle proposé : implantation

Comme nous l'avons mentionné précédemment, nous utilisons un produit peu exigent en termes de lourdeur administrative de la distribution. Nous fonctionnons sur des systèmes d'exploitation *Microsoft*, en se limitant à la gamme NT(2000 et +) de ceux-ci, qui sont plus fiables que leurs prédécesseurs.

Dans ce mode, notre produit apparaît sous la forme d'un service *Windows* en attente de travail à effectuer. Ces modules sont écrits en C++ permettant l'optimisation du code et non sa portabilité qui n'est pas l'objectif de l'étude mais plutôt une preuve d'optimisation de performance dans les architectures distribuées. Plusieurs outils s'offrent à nous pour la répartition réseau, nous n'avons qu'à penser à : Java RMI, CORBA ou. NET de Microsoft pour n'en nommer que quelques-uns. Ces outils permettent tous la prise en charge d'architectures distribuées mais requièrent une utilisation des ressources et du temps processeur qui est assez élevée pour biaiser les résultats de manière significative. C'est pourquoi, nous nous replions sur un modèle qui n'offre pas toute la flexibilité des autres produits, mais qui minimise l'impact sur les résultats obtenus. Nous utilisons donc le plus trivial des modes de communication c'est à dire les « sockets » dans un monde combiné aux services de *Windows*. L'incidence du temps d'accès aux bases de données doit aussi être très limitée, c'est pourquoi, nous utilisons plutôt une base de connaissance en mémoire du répartiteur qui est sur une unité dédiée en accès concurrentiel avec les services de distribution sur les unités de calcul (Figure 1.25).

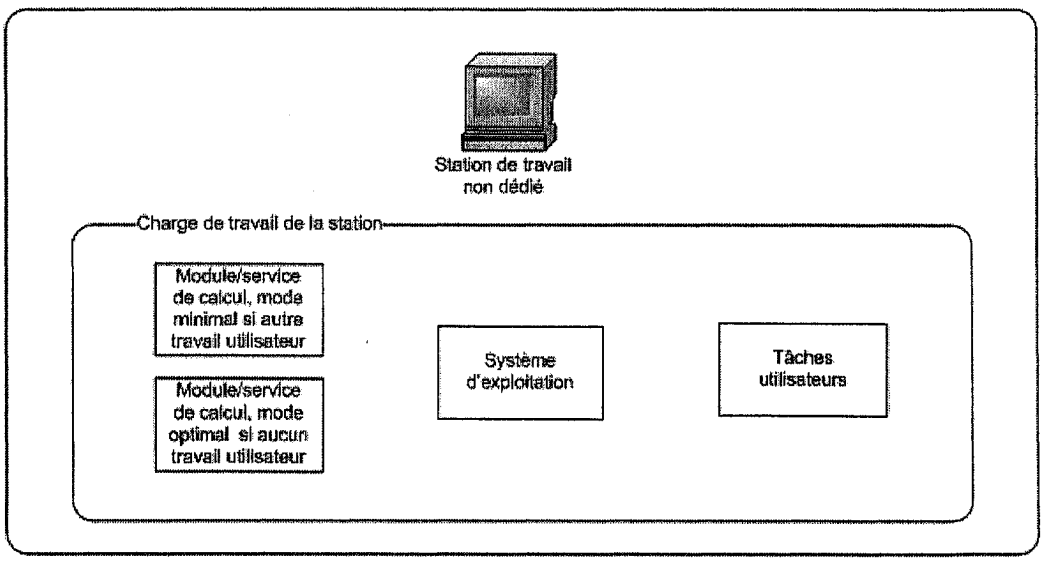

Figure 1.25 La station de travail non dédié.

CHAPITRE 2

MODÈLE PROPOSÉ

2.0 Topologie réseau

Dans le but de regrouper un ensemble d'appareils pour former un centre de calcul capable d'effectuer les différents calculs distribués, il faut adopter une topologie versatile qui permet l'intégration et le retrait de nœuds de calcul et/ou de répartiteurs.

2.1 Les entités de la topologie

2.1.1 Répartiteur

Un module de répartition sert à gérer un sous ensemble de nœuds de calcul et/ou de répartiteurs sous forme d'une arborescence dont les feuilles sont des nœuds de calculs joint par des répartiteurs (Figure 2.1). Chaque répartiteur doit connaître à tout moment la disponibilité des nœuds de calcul dans la branche du sous arbre dont il est la racine.

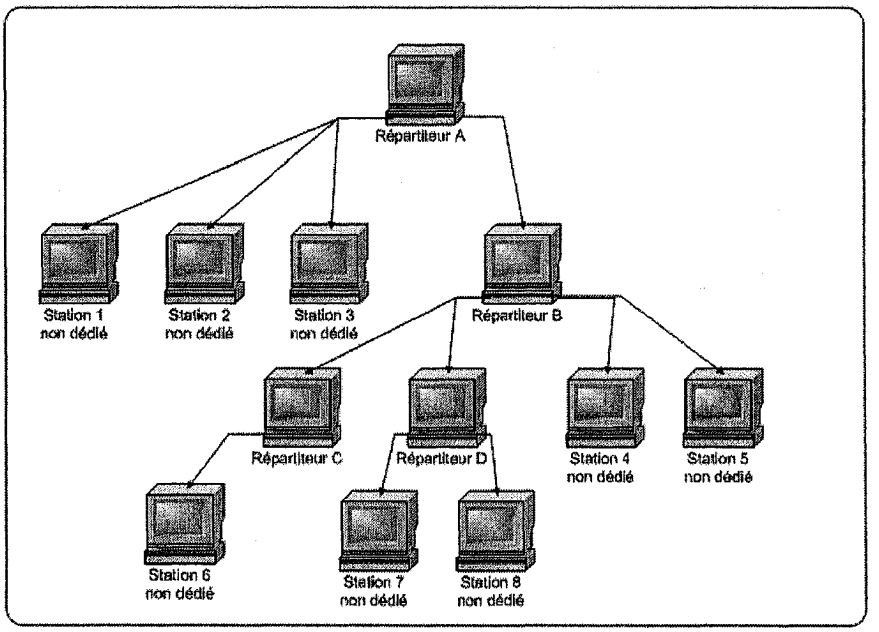

Figure 2.1 Arborescence de répartiteurs.

Voici la liste de connaissances (Tableau I) pour chacun des répartiteurs :

Tableau I

Liste de connaissances

Répartiteur	Stations connues
A	1, 2, 3, 4, 5, 6, 7 et 8
B	4, 5, 6, 7 et 8
C	6
D	7 et 8

2.1.2 Nœud de calcul

Un nœud de calcul est une unité servant à produire un résultat (exécution). Ce nœud doit faire connaître sa disponibilité et sa performance sous forme d'un paquet informatif à un module répartiteur à une fréquence donnée.

2.1.3 Réseau et temps de réponse

Puisque les transferts réseaux sont le goulot d'étranglement en temps de notre système, nous devons minimiser la quantité d'information qui transige par celui-ci. Les paquets d'information échangés entre les nœuds et les répartiteurs doivent être de forme minimale et avoir une fréquence la plus basse possible. Nous appelons « f » le nombre d'envois par seconde. Cependant, la possibilité de retrait des nœuds dans un système non dédié impose une fréquence de base. C'est pourquoi nous devons estimer arbitrairement cette fréquence en fonction du temps de réponse que nous désirons offrir dans les calculs. L'incidence de « f » sur les calculs est étroitement liée à la forme de l'arborescence de répartition et par le fait même ne peut être estimée de manière générale mais simplement estimée particulièrement

pour cet arbre. L'objectif principal de ce travail étant l'optimisation du temps de calcul global, nous devons connaître rapidement la mise en non disponibilité d'un nœud afin de transférer la portion de calcul de ce nœud vers un autre en notant que le temps entamé du calcul ainsi que le délai d'avertissement de l'arrêt du calcul viennent pénaliser notre temps de réponse (Figure 2.3). Afin de ne pas créer d'effet d'oscillation de disponibilité, la fréquence de rafraîchissement des disponibilités du côté des répartiteurs doit être la moitié de la fréquence de transmission des disponibilités des nœuds. Une oscillation de disponibilité de manière générale se produit lorsque la fréquence de mise à jour « Fa » de la valeur de disponibilité par le nœud dépasse la fréquence de lecture d'initialisation « Fb » par le répartiteur (Figure 2.2).

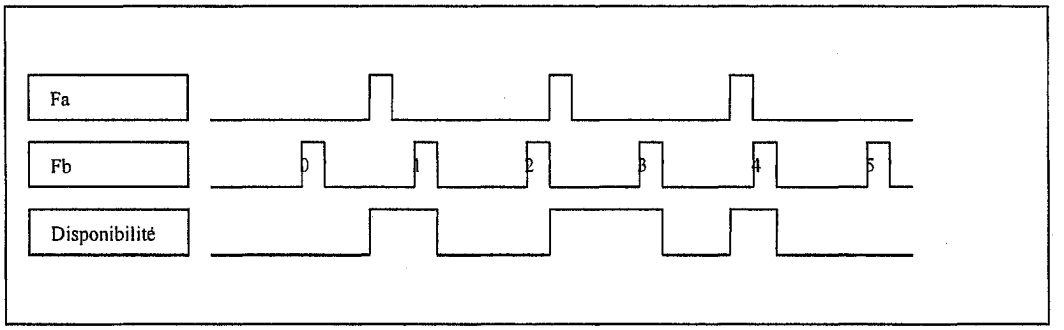

Figure 2.2 Oscillation de disponibilité.

La disponibilité lue (Tableau II) par « Fb » se résume à :

Tableau II

Disponibilité

Intervalle	Valeur lue
0	Non disponible
1	Disponible
2	Non disponible
3	Disponible
4	Disponible
5	Non disponible

Nous constatons alors la fréquence « Fa » doit être supérieure à la fréquence « Fb » et de manière générale, une valeur sécuritaire pour la fréquence de « Fb » est f/2. En conséquence, 2/f devient le facteur temps de pénalité de notre système. Lorsqu'un calcul est interrompu, il existe une période de temps pendant laquelle nous considérons toujours le calcul en cours. Ce temps de mise en non disponibilité varie entre 1/f et 2/f qui est dû à l'implémentation de notre système.

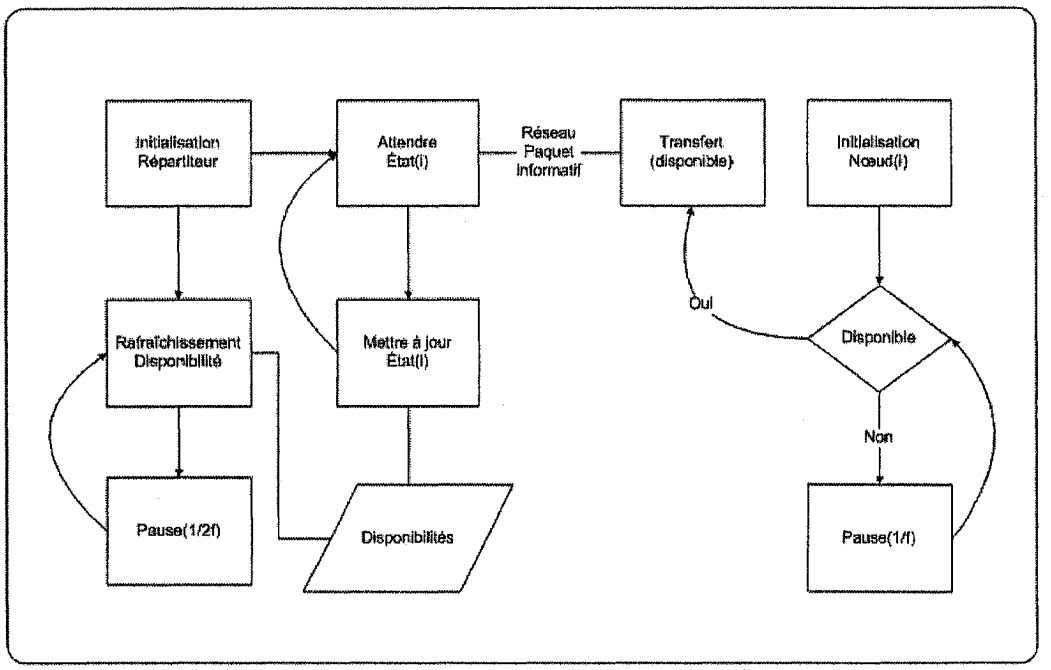

Figure 2.3 Gestion des disponibilités.

Un calcul prend la forme : d'une requête de calcul, du calcul proprement dit et d'un message de réponse (Figure 2.4).

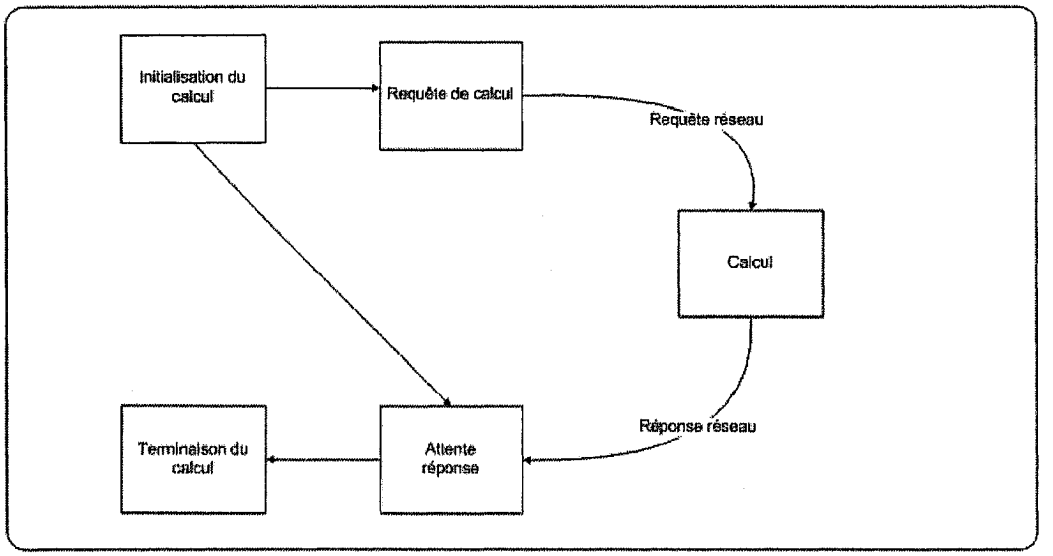

Figure 2.4 Calcul dédié avec disponibilité.

Le temps de calcul dans ce mode est :

$$tempsCalcul_0 = tempsGestion + temps\,\mathrm{Re}quête + tempsCalculNoeud + tempsRéponse$$

Le « tempsGestion» représente le traitement de mise en œuvre indépendant du type de calcul. Il est lié au modèle de répartition et à son mode de gestion, il est présenté comme une constante globale car son influence sur les calculs doit être peu variable. La variance du « temps de gestion » est surtout influencé par l'ampleur de l'arborescence et les différents liens réseaux les unissant.

L'attente de disponibilité en mode dédie (Figure 2.5) ou non dédié (Figure 2.6) ne se calcule pas de manière purement mathématique puisqu'il s'agit là d'un phénomène probabiliste relié à l'utilisation des équipements.

Une étude sur une période donnée de l'utilisation des ressources constituant l'ensemble des nœuds potentiels de calcul doit être faite de manière à créer un historique d'utilisation sur lequel un modèle probabiliste peut être appliqué afin de

déduire l'espérance de disponibilité en temps E d'au moins un nœud pour fin de calcul.

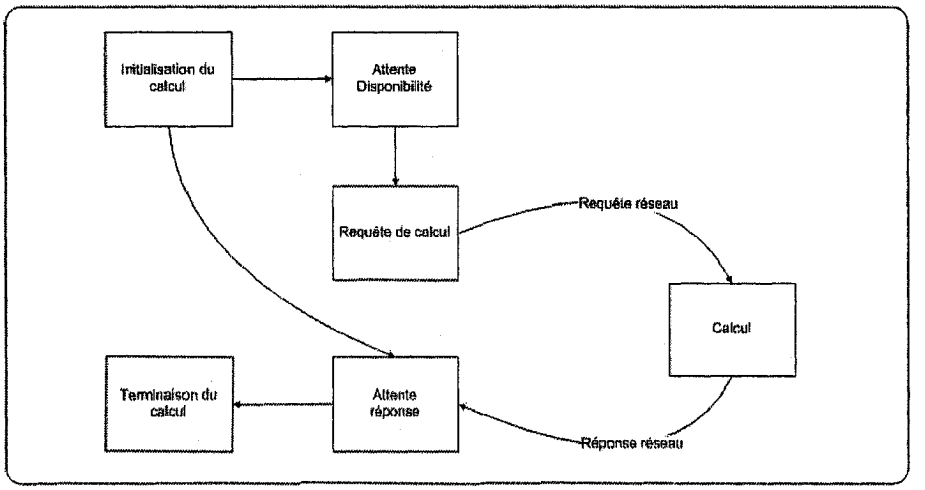

Figure 2.5 Calcul dédié avec non disponibilité.

L'indisponibilité en temps d'un nœud de calcul peut prendre la valeur zéro. Un système avec disponibilité peut être représenté par un système avec non disponibilité dont l'attente est zéro. Le terme de système avec disponibilité n'est donc plus nécessaire et est supprimé dans les références futures de ce document.

Le temps de calcul révisé est :

$$tempsCalcul_1 = tempsGestion + E + temps\,\mathrm{Re}\,quête + tempsCalculNoeud + tempsRéponse$$

Le but de l'utilisation des architectures distribuées est la mise en œuvre réseau d'un mode de calcul parallèle. Les calculs que nous soumettons à notre système sont donc décomposables de manière à ce qu'ils puissent être fait indépendamment de leurs nombres.

La réalisation d'un tel calcul nécessite le déclenchement d'une suite de demandes de calcul fragmenté que nous notons $tempsCalcul_i, 0 \le i \le n$, où n est le nombre de calculs fragmentés du calcul complet.

Le temps total d'un calcul composé en mode dédié revient donc au maximum des temps de calcul distribué, soit :

$$tempsTotal_0 = Max(tempsCalcul_i) + tempsGestion$$

Nous remarquons que le temps de gestion est une valeur non négligeable, mais il peut être estimé de manière générale et il est donc considéré globalement.

Dans le cas de l'arrêt prématuré d'un calcul en mode non dédié (Figure 2.6), le temps de calcul total doit contenir une référence au temps entamé, à la mise en non disponibilité (détection) et à la reformulation du calcul. Le temps de calcul général prend la forme itérative suivante :

$$temps\,\mathrm{Re}\,prise_i^j = tempsEntamé_i^j + tempsDétection_i^j, 0 \le j \le m$$

$$tempsCalcul_i = \left(\sum_{j=0}^{m} temps\,\mathrm{Re}\,prise_i^j \right) + E + temps\,\mathrm{Re}\,quête_i + tempsCalculNoeud_i + tempsRéponse_i$$

La variable « j » représente le nombre de fois que le $Calcul_i$ est interrompu et ce nombre varie entre 0 et m. Ici, « m » peut atteindre une valeur infinie mais cette probabilité est directement liée à « E » et une telle situation ne peut se produire qu'avec un système non dédié dont la disponibilité est très précaire et par conséquent non viable pour une mise en œuvre distribuée. Le temps $E + temps\,\mathrm{Re}\,quête_i + tempsCalculNoeud_i + tempsRéponse_i$ représente le temps de calcul ininterrompu du $Calcul_i$. Notons aussi que la probabilité de l'arrêt de traitement croît directement avec la durée du calcul et que par conséquent les plus grands calculs sont plus sujet à subir une coupure. C'est pourquoi nous devons minimiser ou équilibrer l'écart entre les différents temps de calculs là où c'est possible.

N'ayant pas le contrôle sur la répartition parallèle puisque le système de calculs n'est qu'un transport, cette charge de responsabilité ou prise de conscience relève de l'utilisateur (voir 2.1.4 section Encapsulation et translucidité). Cette notion de temps de réponse vient altérer directement la fréquence d'identification des mises en non disponibilité des nœuds et cette fréquence en relation avec les paquets informatifs vient elle-même engorger le réseau par ces transferts d'information. Nous devons donc établir cette fréquence en relation directe avec la durée moyenne de calcul des nœuds qui ne peut être obtenue que par apprentissage du système au fur et à mesure des différents traitements demandés. Notre fréquence « f » possède un impact direct sur le temps de détection qui varie entre 1/f et 2/f, ou en moyenne 1.5/f sur un large échantillonnage.

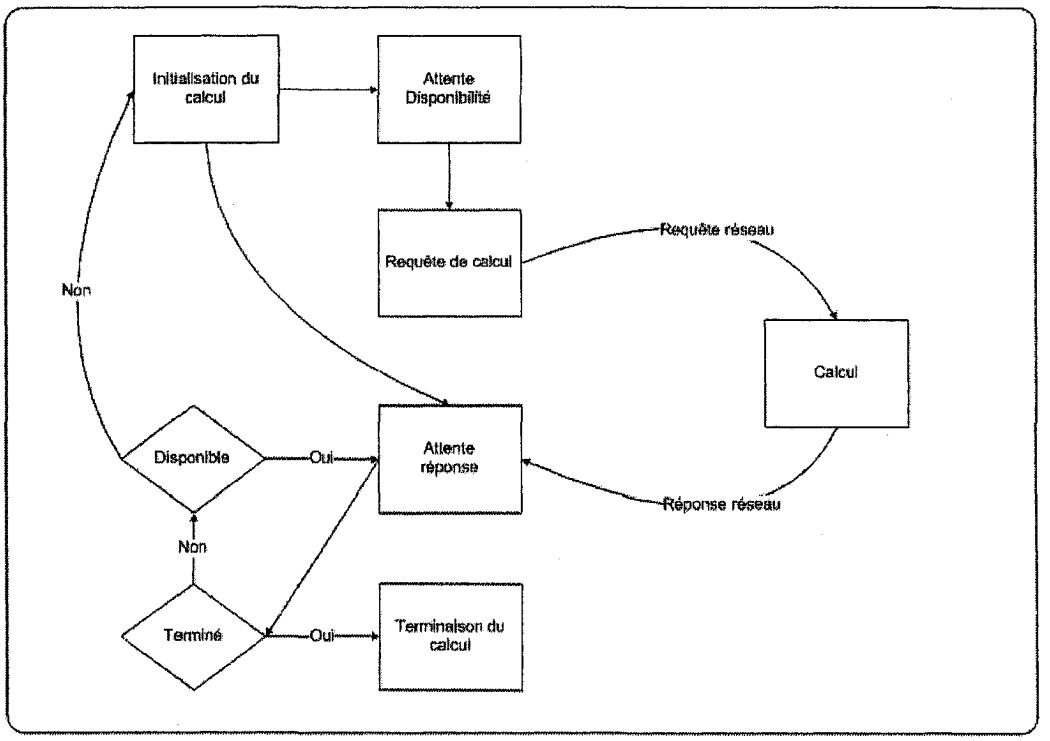

Figure 2.6 Calcul non dédié.

La probabilité qu'un nœud soit disponible assez longtemps pour compléter un calcul est basée sur la variance de disponibilité des nœuds. Les appareils constituants le parc de nœuds et répartiteurs de notre arborescence sont généralement des équipements servants dans des salles publiques dont l'accès n'est que peu contingenté et par conséquent, cette variance n'est que peu prévisible hors mis le fait quelle est grande. Le temps de calcul en mode non dédié est donc basé sur E, l'espérance de disponibilité d'au moins un nœud et V la variance en disponibilité, ce temps peut donc être exprimé de manière générale par ECi, l'espérance en temps de la complétude du $Calcul_i$. L'impact de V sur ECi se fait dans le temps de reprise d'un calcul, plus le calcul dure longtemps, plus il risque d'être interrompu. Nous pouvons minimiser le temps de reprise en dupliquant les mêmes calculs sur plusieurs nœuds, c'est à dire prendre un $Calcul_i$ et le faire effectuer par plus d'un nœud. Le premier nœud qui répond à ce calcul donne la norme de temps pour ce calcul. Le nombre de nœuds nécessaires pour minimiser les temps de reprise varie selon : la durée du calcul (qui n'est pas connu par notre système), la variance en disponibilité V et l'espérance E.

Il est notable qu'en dépit d'une formulation mathématique précise pour cette évaluation, l'ajout d'un nœud supplémentaire peut avoir deux finalités : si le nombre d'appareils disponibles dans notre système dépasse le nombre de calculs à effectuer, ECi diminue, si en contre partie ce nombre est inférieur ou égal au nombre de calculs, ECi croît. En effet, il est évident qu'en situation d'effectif limité en nœuds de calcul, la duplication des calculs ne peut que pénaliser la disponibilité et bien que l'effet de variance V diminue, l'espérance de disponibilité E augmente.

Dans la majorité des cas, nous avons à notre disponibilité un vaste ensemble d'appareils et V est grand. Comme E est très petit la duplication est souhaitable et profitable.

L'établissement du nombre optimal d'appareils pour la duplication n'est cependant pas facile à calculer mathématiquement et il dépend grandement de V et seul un historique important d'utilisation fixe V et implicitement N, le nombre d'appareils optimal pour minimiser ECi. Nous trouvons donc la même formule pour le temps de $Calcul_i$:

$$tempsCalcul_i = \left(\sum_{j=0}^{m} temps\,\mathrm{Re}\,prise_i^j \right) + E + temps\,\mathrm{Re}\,quête_i + tempsCalculNoeud_i + tempsRéponse_i$$

Notons cependant que $\left(\sum_{j=0}^{m} temps\,\mathrm{Re}\,prise_i^j \right)$ tend vers zéro et que le phénomène de duplication (Figure 2.7) redresse le temps de calcul vers un modèle dédié. Cette dernière remarque est importante car nous avons un système non dédié qui par son nombre d'appareils produit un résultat en temps égal à un système dédié et optimisant ainsi l'utilisation des équipements à notre disponibilité.

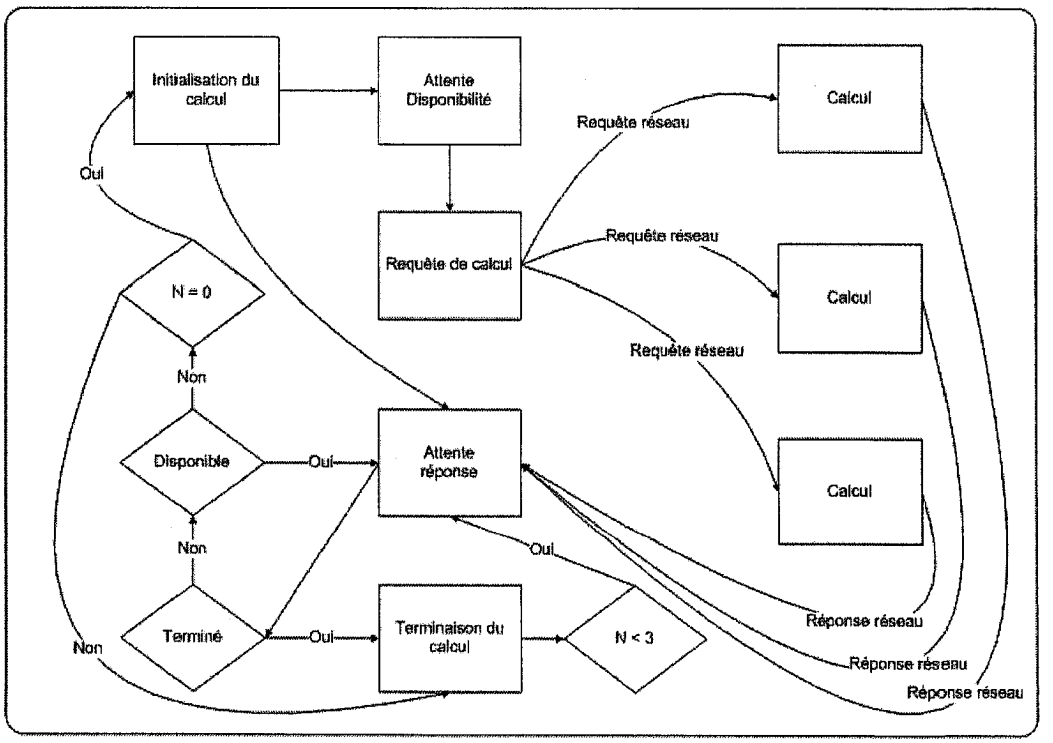

Figure 2.7 Duplication des calculs (N=3).

Le terme conditionnel « Disponible » concerne la disponibilité globale et seul l'absence totale de disponibilité c'est à dire tous les nœuds indisponibles, produit une demande de réinitialisation du calcul qui est redistribué vers un autre ensemble de N nœuds. L'attente totale des réponses est prolongée au-delà de la réception d'une première réponse valide, afin de conclure convenablement le mode dupliqué. ECi n'augmente pas car le calcul se continue dès la réception de la première réponse valide.

Cet algorithme minimise la sommation des temps de reprise qui ne se produisent que si l'ensemble de N nœuds devient indisponible dans sa totalité. Si les reprises sont remarquées régulièrement, N peut être révisé dans notre système et fixé comme nouveau paramètre d'exécution de notre arborescence.

2.1.4 Encapsulation et translucidité

Notre système sert de transport pour fin de calcul, il met en disponibilité un regroupement de nœuds de calcul sous forme d'arbre pour un demandeur de service. Le seul point de contact entre le demandeur et l'arborescence est un répartiteur, cette encapsulation rend l'utilisation du système simple et translucide. Le demandeur n'a jamais à connaître la structure de l'arborescence, mais simplement un point de contact et le protocole de demande/retour de service sous forme de calcul. L'interface entre les deux mondes doit être optimisée afin de réduire les latences de communication entre les deux parties. L'utilisation du système se résume à une suite séquentielle de demandes de calcul asynchrone identifié par une fragmentation initiale du problème par l'utilisateur. Dans ce mode asynchrone, les réponses arrivent dans un ordre qui peut différer de l'ordre de demande et l'identification des calculs sert à l'utilisateur pour reconstituer le résultat.

2.1.5 Minimisation des transferts

Nous avons précédemment discuté de l'efficacité globale de traitement versus les temps de transfert et nous avons noté que les transferts étaient le centre critique de ce processus. C'est pourquoi nous devons accentuer nos efforts sur la minimisation des transferts réseaux. La quantité de données échangées au travers de notre arborescence doit être minimisée afin de diminuer le temps global de calcul. Notre système offre des performances accrues sur des calculs ne prenant que peu de données pour leurs initiations (en entrée), c'est pourquoi nous considérons ces temps de transfert d'information à la demande comme largement inférieurs au temps de transfert des résultats du calcul. Dans notre arborescence, les informations d'une demande suivent une route entre la demande initiale au répartiteur interface (la racine d'un sous arbre de notre système) et le nœud de calcul où le traitement est effectué. Ce chemin est donc emprunté pour toute information échangée entre le demandeur (client) et le nœud de calcul (serveur).

Le chemin entre le client et le serveur peut être direct ou plus ou moins complexe en fonction de la formation de notre arborescence (Figure 2.1). Chaque information échangée entre les répartiteurs de notre système est répétée (logiciel et matériel) sur le(s) réseau(x) et provoque un ralentissement notable augmentant ainsi le temps global de calcul. Nous pouvons donc voir une possibilité d'optimisation logicielle des transferts en utilisant une réponse directe entre le serveur et le client. Cette réponse directe n'affecte en rien la translucidité de notre système puisque que le demandeur utilise une interface de communication avec le répartiteur, il ne sait donc pas d'où physiquement provient la réponse.

2.1.6 Réseau privé

Le chemin de demande dans notre système peut passer par des segments de réseaux privés qui, par sa définition, n'offre pas de chemin direct entre le demandeur et le nœud de calcul.

Si ce réseau privé est doté d'un appareil avec une vue sur la partie privée et une vue sur la partie extérieure des deux réseaux, il est possible de mettre un module répartiteur sur cet appareil et ainsi de former un pont de calcul pour notre arborescence entre la section privée et la section extérieure du réseau (Figure 2.8). La création de réseaux privés vise essentiellement à cacher certains équipements du monde extérieur et ainsi limiter leurs accès. Il est cependant fréquent que les appareils du réseau interne (privé) aient besoin d'avoir un accès vers l'extérieur (ex : Internet) par le biais d'une passerelle de sortie, ce qui limite l'accès entrant du réseau et permet une sortie vers le monde extérieur. Dans ce mode, le chemin client/serveur doit passer par un répartiteur pont, mais la réponse serveur/client peut de manière logicielle être acheminée directement. Il faut toutefois noter que l'utilisation d'un répartiteur pont introduit une faille de sécurité au niveau de l'accès extérieur aux ressources internes.

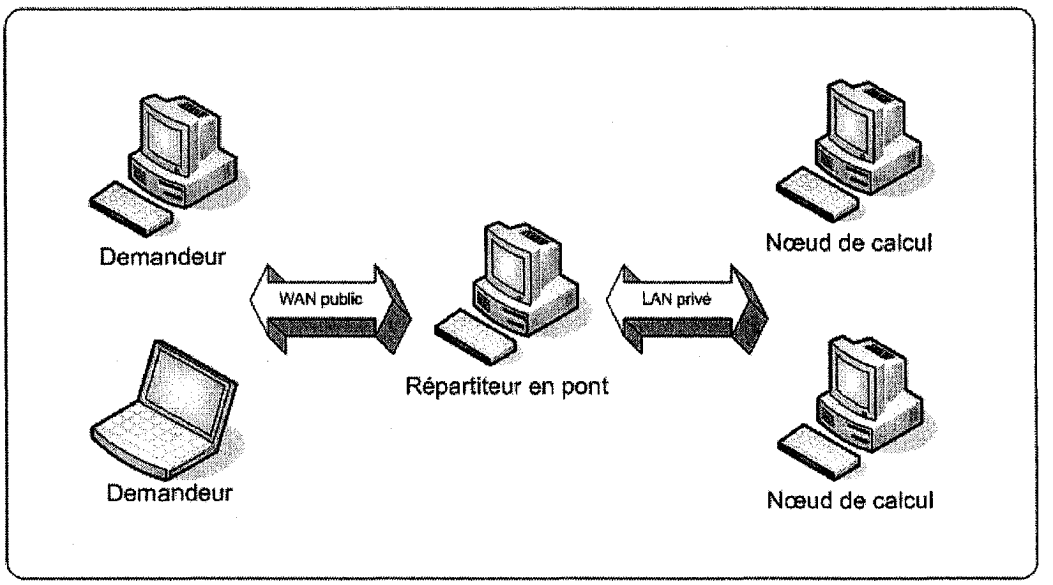

Figure 2.8 Répartiteur en pont.

Tout répartiteur de l'arborescence peut devenir un « Répartiteur pont » et le réseau peut contenir plusieurs sous réseaux privés.

2.1.7 Protocole de communication dans l'arbre

La flexibilité de notre système aux variances de disponibilités des nœuds et répartiteurs ne doit pas influencer la complexité de la gestion de cette arborescence, c'est pourquoi les nœuds et répartiteurs (les entités de nôtre système) doivent se reconnaître de manière transparente. Pour communiquer entre elles, les entités utilisent des paires adresses IP/Port que nous notons « Point de communication IP (PCIP)» et à toutes fins pratiques, une entité ne doit connaître dans l'arborescence que l'entité au dessus d'elle. Cette connaissance limitée mais récursive permet la construction de notre arbre. Nous notons « maître » une entité supérieure PCIP dans l'arborescence ou encore une entité nous rapprochant d'un niveau sur le chemin menant vers la racine de l'arbre et notons « local » l'identification PCIP du nœud lui-même. Toute entité de notre système a donc la connaissance d'un maître sauf pour un nœud qui est la racine de notre arbre. Le nœud n'ayant pas de maître est la racine de notre arbre (Figure 2.9). Le maître ne peut être qu'un répartiteur car les nœuds ne gèrent qu'une unité de calcul et ne fait pas la gestion des ressources qui est le rôle des répartiteurs.

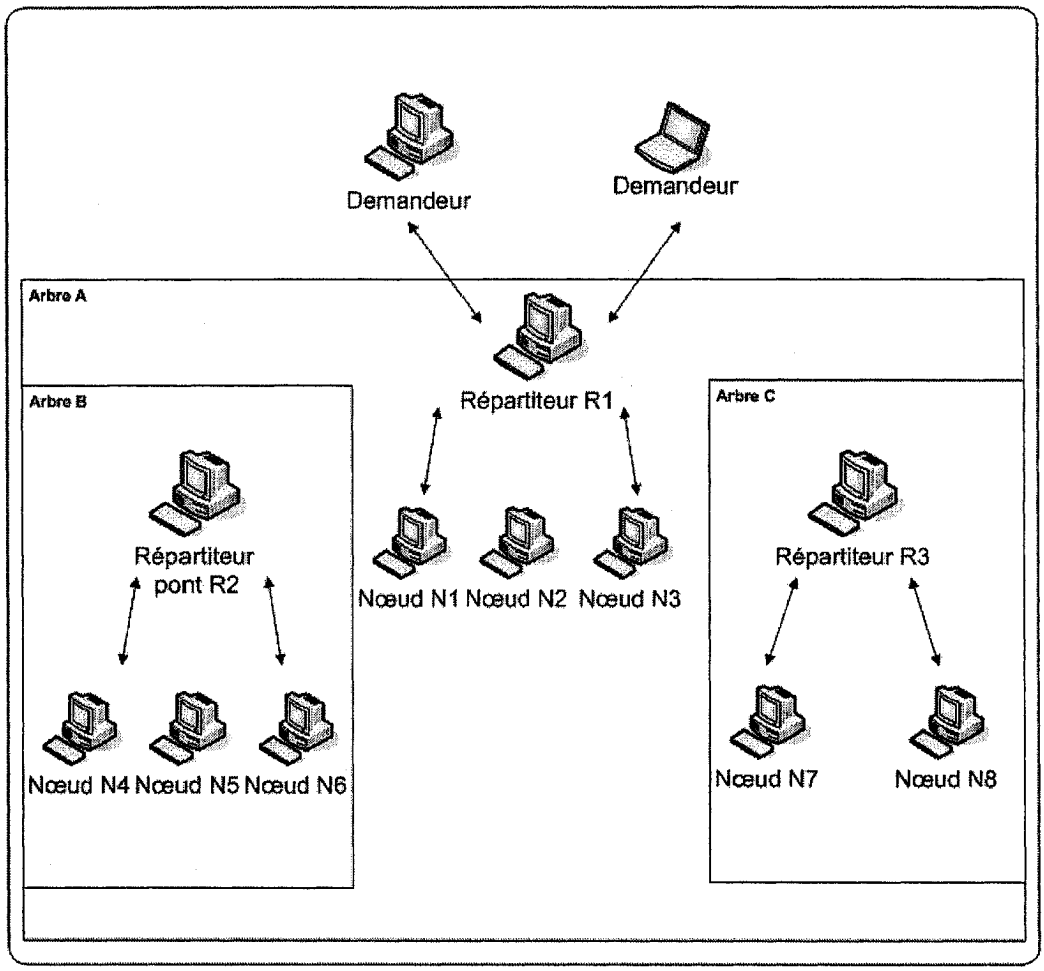

Figure 2.9 Arborescence et PCIP.

Chaque nœud de calcul qui est une feuille dans notre arbre (N1-8, Figure 2.9), doit signifier sa présence à une fréquence donnée par un paquet informatif, celui-ci transige dans l'arbre, de la feuille vers son maître permettant ainsi la connaissance de sa disponibilité (N1-3->R1, N4-6->R2, N7-8->R3). Si ce maître possède lui-même un maître, il doit faire connaître la disponibilité à celui-ci et de manière récursive jusqu'à la racine (R2-3->R1). Ce cheminement crée des ensembles grandissants de disponibilités au fur et à mesure de l'approche de la racine.

Tableau III

Connaissances

Répartiteur	Connaissance Descendante	Connaissance Ascendante
R2	N4 N5 N6	R1
R3	N7 N8	R1
R1	N1 N2 N3 R2 R3 N4 N5 N6 à travers R2 N7 N8 à travers R3	Aucune (racine de l'arborescence)

Cette structure de disponibilité permet la reconstruction inverse du cheminement dans l'arbre (racine vers nœud) par la connaissance en sous ensemble d'identification de la provenance. A des fins pratiques, nous avons besoin de connaître l'identification d'un nœud par la paire constituée de son PCIP propre et le PCIP d'où il provient, si cette connexion est directe, l'identification sera le doublé PCIP propre.

Tableau IV

Paires d'identification

Répartiteur	Ensembles des paires d'identification
R2	(N4, N4) (N5, N5) (N6, N6)
R3	(N7, N7) (N8, N8)
R1	(N1, N1) (N2, N2) (N3, N3) (N4, R2) (N5, R2) (N6, R2) (N7, R3) (N8, R3)

Cette redondance n'a lieu qu'à l'identification initiale du nœud de calcul à son maître répartiteur, sinon, la paire d'identification est constituée du PCIP du nœud et du PCIP du répartiteur relais que nous notons « passerelle de provenance » ou

simplement « passerelle ». Ce terme ne se rapporte qu'aux répartiteurs de notre système et n'est que protocolaire puisqu'il ne crée pas de nouvelles formes (autres que noeuds de calculs et répartiteur), mais n'ajoute qu'une fonctionnalité dans nos modules de répartition.

Pour communiquer d'un répartiteur vers un nœud de calcul, nous adressons le nœud par le PCIP de sa passerelle qui au dernier moment de transition sera le PCIP du nœud lui-même. Imaginons simplement un scénario dans lequel un nœud se connecterait à l'arbre par un premier répartiteur « R2 » de maître « R1 ». La paire identification du nœud « N4 » dans l'arbre variera comme suit :

Tableau V

Exemple de transition

Répartiteur	Paire identification
R2	(N4, N4)
R1	(N4, R2)

Notons que « R1 » est la racine de notre arbre.

- Pour aller de « R1 » vers « N4 », il faut passer par « R2 » qui est dans la première paire identification
- Pour aller de « R2 » vers « N4 », nous contactons « N4 » directement par la paire identification.

De cette manière l'algorithme est transparent au système. Puisqu'il ne va que de passerelle en passerelle. La dernière passerelle est le nœud lui-même.

Comme nous l'avons mentionné précédemment, chaque répartiteur de notre arbre connaît la liste des disponibilités totales de son sous arbre par une liste de « paire identification » des nœuds.

Prenons un exemple avec une profondeur d'arborescence supérieure (Figure 2.10). L'ensemble des paires d'identification suivant identifie la connaissance de chacun des répartiteurs à travers notre arborescence.

Tableau VI

Exemple de connaissance à 4 niveaux

Répartiteur	Ensembles des paires d'identification
R4	(N4, N4)
R3	(N3, N3) (N4, R4)
R2	(N2, N2) (N3, R3) (N4, R3)
R1	(N1, N1) (N2, R2) (N3, R2) (N4, R2)

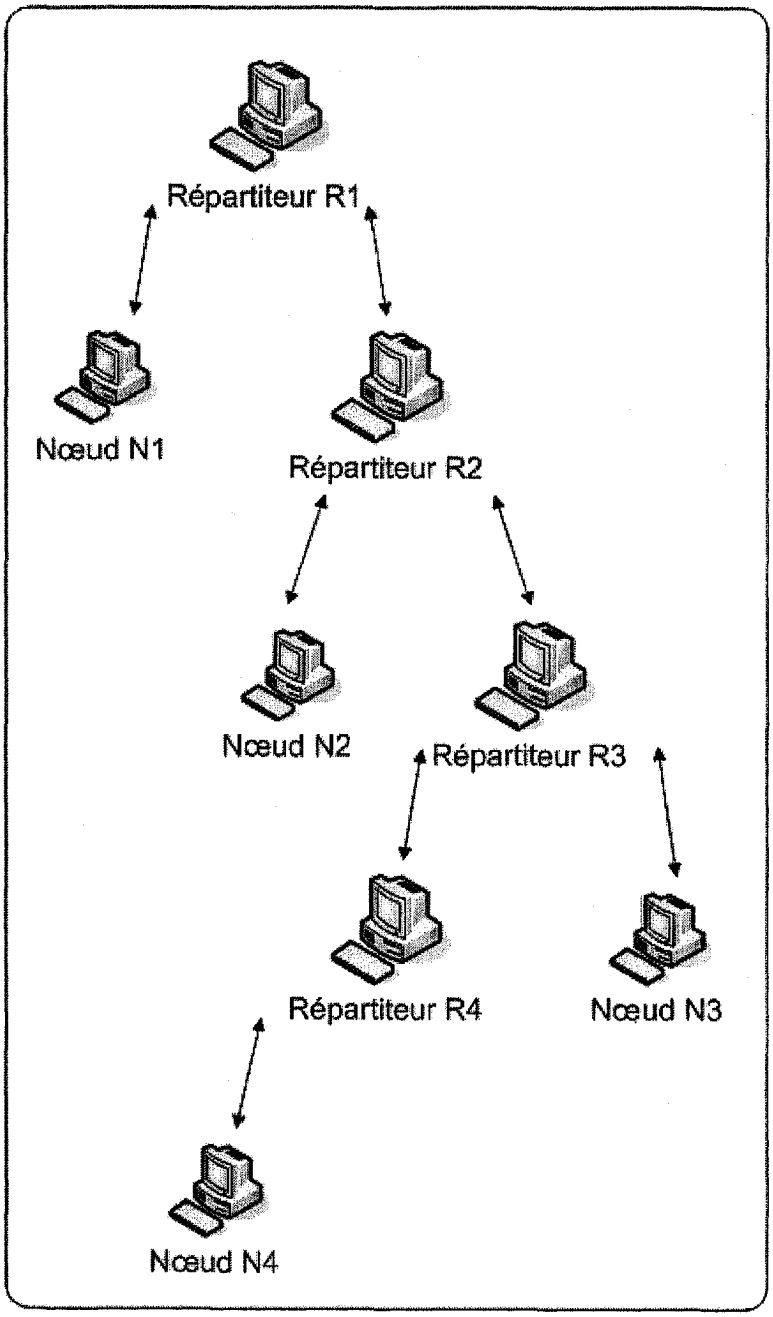

Figure 2.10 Arborescence PCIP et cheminement.

Nous obtenons de manière générale, l'ensemble de connaissances global suivant :

Tableau VII

Connaissance Ascendante et descendante

Répartiteur ou Nœud	Connaissance Descendante	Connaissance Ascendante
N4	Aucune, feuille de l'arbre	R4
R4	N4	R3
N3	Aucune, feuille de l'arbre	R3
R3	R4, N3, N4	R2
N2	Aucune, feuille de l'arbre	R2
R2	R3, N2, R4, N3, N4	R1
N1	Aucune, feuille de l'arbre	R1
R1	R2, N1, R3, N2, R4, N3, N4	Aucune, racine

Lors d'une communication entre « R1 » et « N4 », à toute fin pratique, nous ne devons connaître que la destination et sa passerelle initiale, l'algorithme récursif fait le travail de transition au long du chemin « R1 » vers « N4 », nous obtenons le parcours suivant :

Tableau VIII

Passerelles et chemins

Étape du parcours descendant, requête	Adressage par paire d'identification
R1->R2	(N4, R2)
R2->R3	(N4, R3)
R3->R4	(N4, R4)
R4->N4	(N4, N4)

Les répartiteurs de notre système agissent à titre de répétiteurs pour la direction des requêtes vers les nœuds et de manière générale, pour les réponses. La seule contrainte pour le chemin des réponses est que la transition doit se faire vers les maîtres de l'arbre en remontant vers la racine requérante qui peut ne pas être la racine de l'arbre, mais simplement un répartiteur (voir Interface au maillage).

Un protocole s'impose pour le cheminement des requêtes/réponses de notre système, il prend la forme minimale composée d'un quadruplet d'adressage comprenant :

- PCIP origine (demandeur)
- PCIP retour (demandeur cascade)
- PCIP distant (nœud)
- PCIP direct (passerelle)

PCIP origine : c'est le répartiteur qui fait la demande initiale de service

PCIP retour : c'est le dernier répartiteur par lequel la requête a transitée.

PCIP distant : c'est le nœud vers lequel le calcul ou service est demandé.

PCIP direct : c'est la passerelle qui sert d'approche et ultimement de contact avec le nœud de calcul.

L'exemple d'une requête du répartiteur « R1 » vers le nœud de calcul « N4 » dans le modèle trivial (Figure 3.2) produit l'évolution du quadruplet du tableau 9.

Tableau IX

Protocole de cheminement

L'entête au niveau de l'entité	Origine	Retour	Distant	Direct
R1	R1	R1	N4	R2
R2	R1	R2	N4	R3
R3	R1	R3	N4	R4
R4	R1	R4	N4	N4

La communication dans ce modèle se fait toujours du PCIP retour vers le PCIP direct, nous voyons ainsi le cheminement de notre requête tout au long de l'arbre, de « R1 » vers « N4 ».

Le chemin de retour de la réponse prends la forme suivante:

Tableau X

Évolution du protocole au retour

Entité	Origine	Retour	Distant	Direct
N4	N4	N4	R1	R4
R4	N4	R4	R1	R3
R3	N4	R3	R1	R2
R2	N4	R2	R1	R1

La communication se fait toujours du PCIP retour vers le PCIP direct.

L'algorithme de transfert doit cependant connaître le type de transport qui peut être soit une requête ou une réponse. Il faut donc ajouter à l'entête du message constituant une requête ou une réponse, un champ qui contient le type de message envoyé, ainsi un paquet informatif et une réponse auront le même comportement à une destination près.

2.1.8 Mise à jour de l'arborescence

Dans un système non dédié, la disponibilité ne relève pas de la gestion de l'arborescence. L'ajout et le retrait doivent être simplifiés pour ainsi maintenir de manière transparente à l'utilisateur la structure de distribution des calculs. Il n'y a que deux types fondamentaux dans notre arborescence : le répartiteur et le nœud. Le nœud signalise sa présence avec l'envoi d'un paquet informatif vers le seul point connu pour lui de l'arborescence c'est à dire un et un seul maître qui est un répartiteur. Le répartiteur est lui responsable du sous arbre dont il est racine et il doit signaliser sa présence à zéro ou un seul maître qui est lui-même répartiteur. Chaque répartiteur a un maître, sauf pour la racine de l'arborescence totale qui lui, n'en a pas (Figure 2.11).

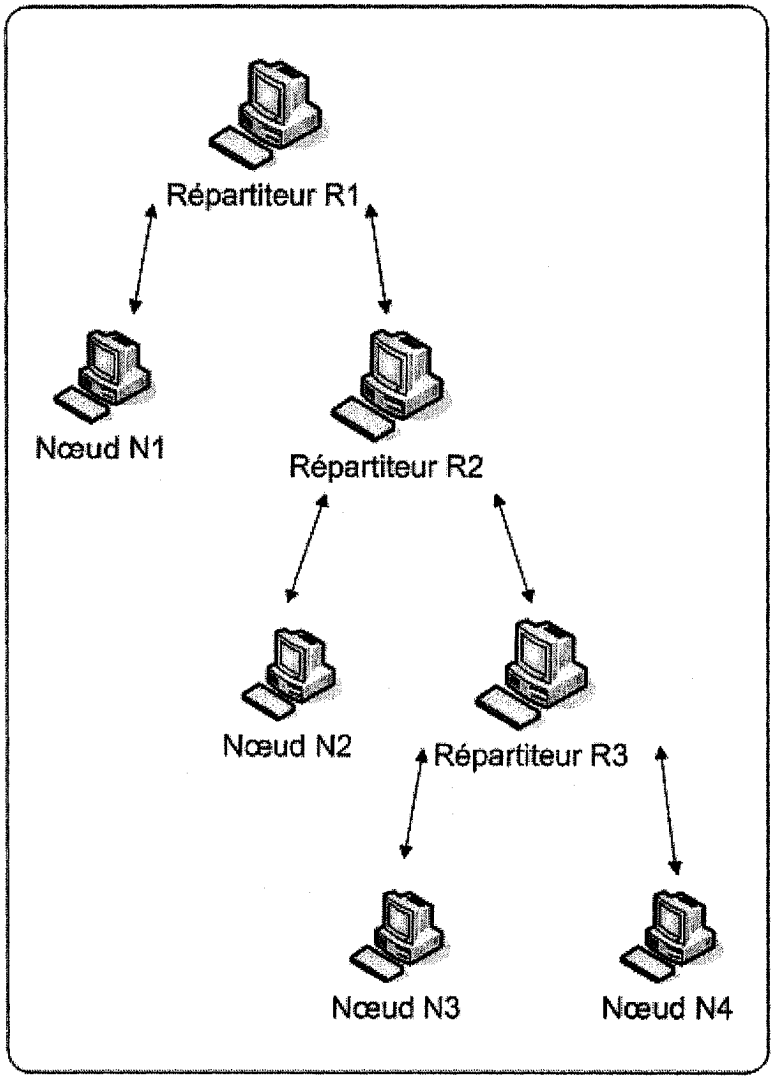

Figure 2.11 Répercussion dans l'arborescence.

L'ajout d'un nœud à l'arborescence peut soit augmenter le nombre de nœuds disponible de un, ou dans le cas ou le répartiteur maître est inactif, ne produit aucun effet. Les nœuds tentent de mettre à jour leur état vers leur maître à la fréquence f, dans le cas ou le maître n'est pas présent, l'opération se termine par une erreur de communication et le nœud continue d'opérer de manière normale (Figure 2.12).

L'ajout d'un répartiteur augmente l'arborescence totale du sous arbre de ce répartiteur et les nœuds qui étaient en absence de maître s'inscrivent dans la période 1/f suivante. Le répartiteur achemine les mises à jour d'états des nœuds vers son propre maître de manière synchrone à leur réception, dans le cas ou le maître n'est pas présent, l'opération se termine par une erreur gérée de communication et le répartiteur continue d'opérer de manière normale (Figure 2.13).

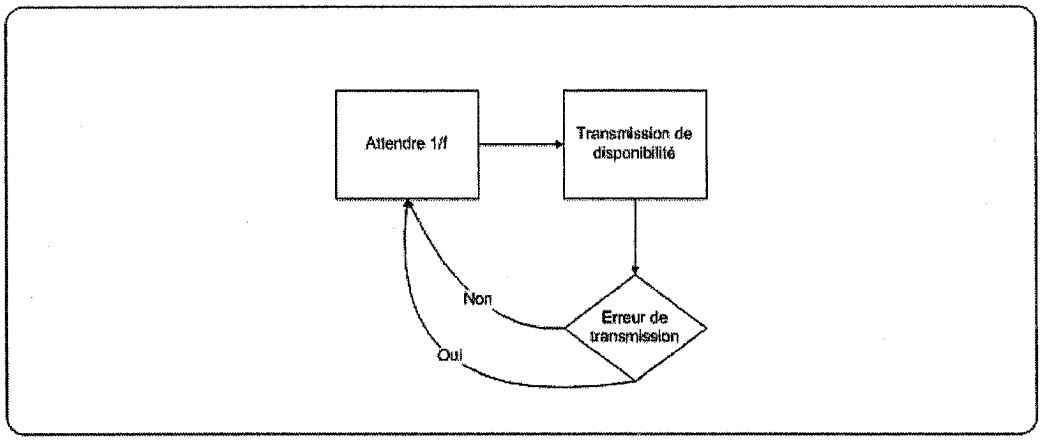

Figure 2.12 Processus de transmission des paquets informatifs pour un nœud.

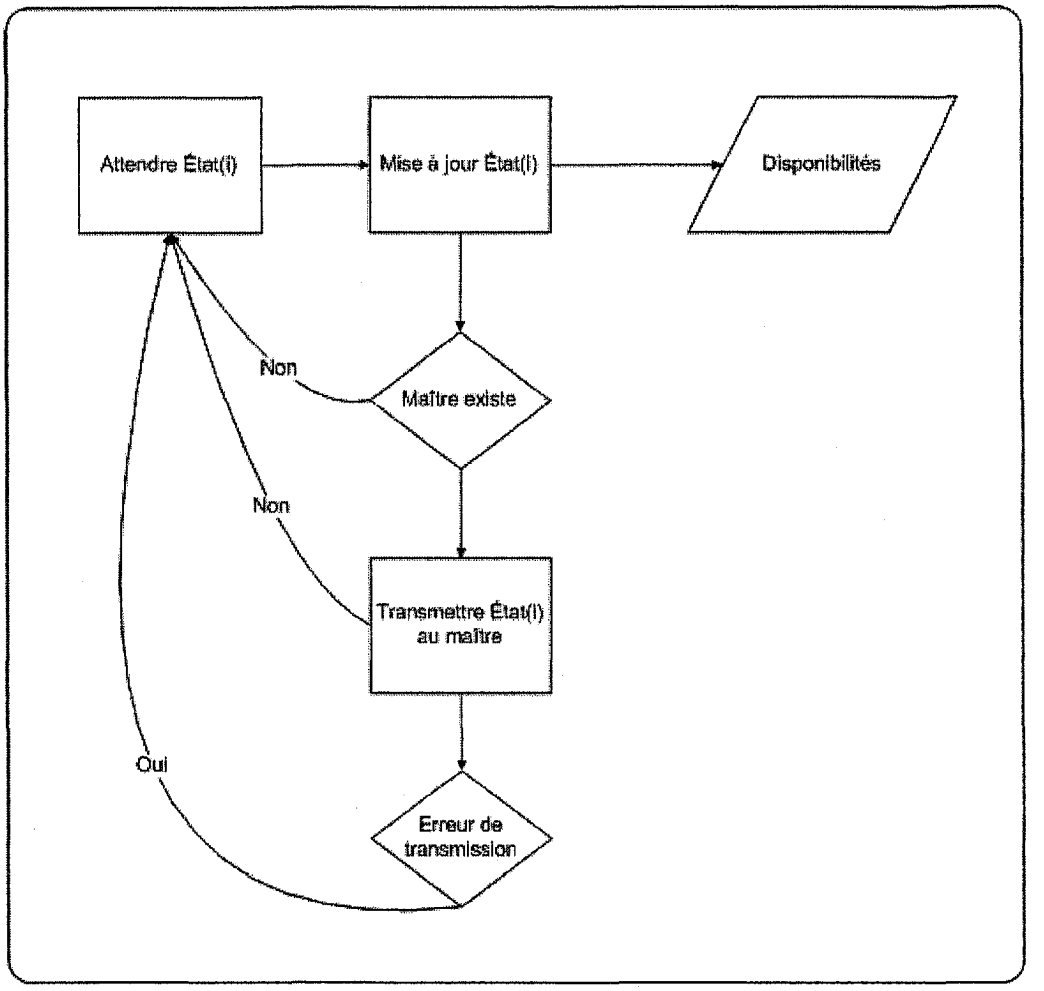

Figure 2.13 Cheminement des disponibilités pour un répartiteur.

Examinons le cheminement du paquet informatif du Nœud N4 dans la figure 2.11. D'abord N4 signalise sa présence avec la paire identification (N4, N4) à R3 qui met à jour sa table de disponibilités et ensuite envoi un paquet informatif de paire identification (N4, R3) à R2 qui met à jour sa table et retransmet un paquet informatif de paire identification (N4, R2) à R1 qui met à jour sa table et termine le cheminement de disponibilité de N4 dans l'arborescence.

Il peut arriver des cas ou les nœuds sont placés hors tension, alors le processus de vieillissement de 2/f retire les disponibilités de ces nœuds.

Si un répartiteur est mis hors tension, alors le processus de vieillissement retire l'ensemble des nœuds qui dépendent de ce répartiteur. Dans la figure 2.11, le retrait de R3 entraîne la suppression par vieillissement de N3 et N4 en R2 et R1.

L'ajout d'un nouveau nœud N5 en R3, de manière transparente ajoute ce noeud N5 en R3, R2 et R1. Cette flexibilité est nécessaire afin de ne pas avoir à perpétuellement reconstruire une arborescence qui serait considérée comme statique, elle rend donc notre arborescence totalement dynamique aux ajouts et suppressions.

2.2 Sécurité

Un tel système, dû à sa versatilité de communication entre PCIP est cependant sujet à recevoir des connexions illicites. Les réseaux actuels reçoivent ce que l'on nomme des « scans » permettant à des programmes dit « robot » de traquer les faiblesses potentielles des réseaux et permettre ainsi l'intrusion sur les appareils d'un réseau. Il est donc impératif d'identifier le demandeur de connexion afin de pouvoir valider ou refuser la connexion en tenant compte de ne pas perdre la flexibilité qu'offre notre système tant au retrait qu'à l'ajout translucide d'entités à l'arbre. Pour résoudre ce problème, nous devons introduire des clés d'encodage connues par les entités de notre système qui permettent de crypter l'information transmise.

Lorsque nous parlons de flexibilité d'ajout et de retrait d'entités à l'arbre ou aux arbres, ce n'est pas sans avoir installé localement sur un appareil, un module double fonctions répartiteur/nœud dont les fonctionnalités séparées peuvent être activées ou inhibées. Lors de cette installation, il faut fournir la clé d'opération (clé d'encodage) de l'arbre. Une demande de connexion IP standard donne une certaine quantité d'information sur le demandeur c'est à dire son IP et le port par lequel il communique. Cette information doit coïncider avec la partie retour de l'entête énumérée dans la section « Protocole de communication dans l'arbre ». Cette comparaison nous indique la validité de la connexion. Si celle-ci diffère, la demande est rejetée et la ligne de communication « socket » est aussitôt fermée.

Toute l'information transférée est encodée par la clé d'opération de l'arbre et seul un module de même clé peut reconstituer logiquement cette information.

Le but de cette œuvre n'est pas de créer un algorithme d'encodage indéchiffrable mais de limiter facilement et efficacement l'accès aux ressources de notre arbre de ressources. Il s'agit d'un encodage sur clé avec un nombre de « bits » assez élevé pour ne pas tomber sur la solution par hasard. De plus, afin d'éviter de reconnaître la structure de transfert, le jeton d'encodage est basé sur le temps avec un double encodage qui permet à des paquets successifs de ne pas être identiques (Figure 2.14).

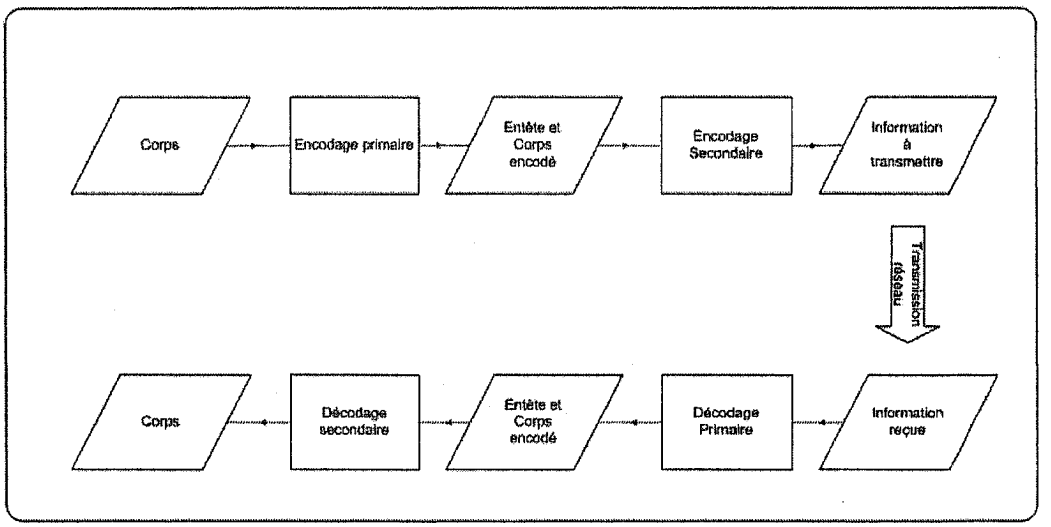

Figure 2.14 Procédé d'encodage et décodage.

Il faut tenir compte du temps que prend l'encodage de toute les données transmises et comprendre que la complexité de l'algorithme d'encodage influence directement le temps minimal de réponse de notre système. C'est pourquoi, cet algorithme assez trivial de double encodage temporel élimine les erreurs de « scans » en coupant la connexion et dissuade une grande partie des « hackers » potentiels de notre système en minimisant l'impact sur le temps de calcul.

Le corps du message est soit la requête ou la réponse à une requête qui voyage dans l'arborescence entre un répartiteur et un nœud de calcul. Il contient le type de message et l'information à joindre à cette transmission.

L'entête est constituée du : PCIP, d'une clé de hachage résultant de l'encodage du corps et d'une variante temporelle connue aux entités participant à cette échange. De sorte que si un même paquet d'information est envoyé plusieurs fois sur le réseau, sa forme transmise est différente. Il est donc plus difficile d'analyser ou déduire la méthode d'encodage. Il est cependant à noter que la partie encodée correspondante à l'entête ne varie que sur la sous section de la variante temporelle, c'est une faiblesse de moindre impact qui est partiellement paliée par une distribution non séquentielle de l'entête à travers le message transmis.

2.3 Interface de requête

Le système à été crée dans le but d'être le plus possible transparent d'utilisation par des primitives d'appels simples et efficaces. Deux grands types de requête sont possibles dans notre système, il s'agît des requêtes natives et non natives.

2.3.1 Requête native

Une requête native est composée dans un langage de programmation compatible à celui des modules de répartition et peut ainsi se résumer à un appel de méthodes dynamiques après l'ajout d'une référence de classe là où le langage le permet. Dans ce cas, une librairie « assembly » est disponible (C#) et permet par des primitives de bases d'effectuer les requêtes aux nœuds dans notre système. Nous rappelons que le système ne sert que de transport aux requêtes/réponses et que par conséquent, les modules de calcul sont fournis par l'utilisateur. Dans le cas de librairies natives, elles doivent respecter certaines contraintes que nous avons minimisées. Il ne faut en aucun cas, être obligé de reconstruire l'arbre ou ses entités pour effectuer un changement de calcul, il faut donc que le transporteur ignore ce qu'il transporte. Le tout est effectué par une « surclasse » de transport que nous notons « Objet Distant (OD) ». L'utilisateur fournit les librairies natives de calcul qui résident sur chacun des nœuds devenant ainsi disponibles au calcul personnalisé. Nous montrons la méthode de distribution dans la section « 2.3.1.7 Distribution des librairies de calcul ».

Afin d'invoquer ces librairies distantes ou classes distantes, nous utilisons l'OD par l'intermédiaire d'une autre « surclasse » que nous notons « Méthode distante (MD) ». La MD permet de définir l'OD sur lequel nous désirons effectuer le travail. Le transporteur n'achemine qu'un objet de la classe MD générique ce qui lui permet de ne pas connaître ce qu'il transporte et ainsi obtenir la translucidité (encapsulation) recherchée, l'objet réel n'est connu qu'aux extrémités par le module appelant et la librairie exécutante (Figure 2.15). Le « nœud coquille » et « nœud interne » représente un nœud, mais subdivise celui-ci en niveaux fonctionnels entre le traitement d'interface (coquille) et l'exécution proprement dite par segments exécutifs de l'appel des fonctions ou méthodes utilisateurs qui résident sur le nœud.

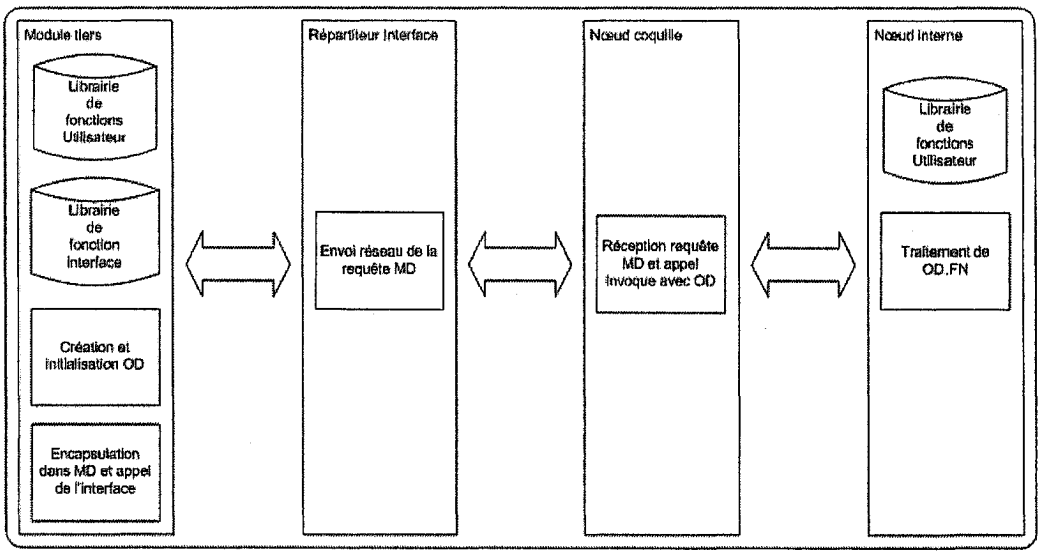

Figure 2.15 Encapsulation.

2.3.1.1 Objet Distant (OD)

L'utilisateur crée les fonctions de calcul sous forme de librairies de classes qui sont distribuées sur les nœuds de calcul. Pour que le transporteur ne connaisse pas ce qu'il transporte, l'OD contient une référence sur la classe et la méthode à invoquer (Figure 2.16). La forme de l'OD est décrite :

Tableau XI

Description OD

Fn
Par
Rep
Nom DLL
Nom classe
Obj

- Fn : représente un pointeur sur une fonction (méthode) à appeler dans une librairie utilisateur située sur un nœud.
- Par : de type objet abstrait, il s'agît du paramètre d'appel de (Fn), le type objet abstrait permet d'y placer un tableau de paramètres si nécessaire.
- Rep : de type objet abstrait, il s'agît de la réponse de l'appel de (Fn), le type objet abstrait permet d'y placer un tableau de réponses si nécessaire.
- Nom DLL : Une chaîne de caractères représentant le nom que porte la librairie utilisateur de calcul se trouvant sur les nœuds.
- Nom Classe : Une chaîne de caractères représentant le nom de la classe contenant « Fn », la méthode distante à invoquer.
- Obj : Un objet valide de la classe « Nom Classe »

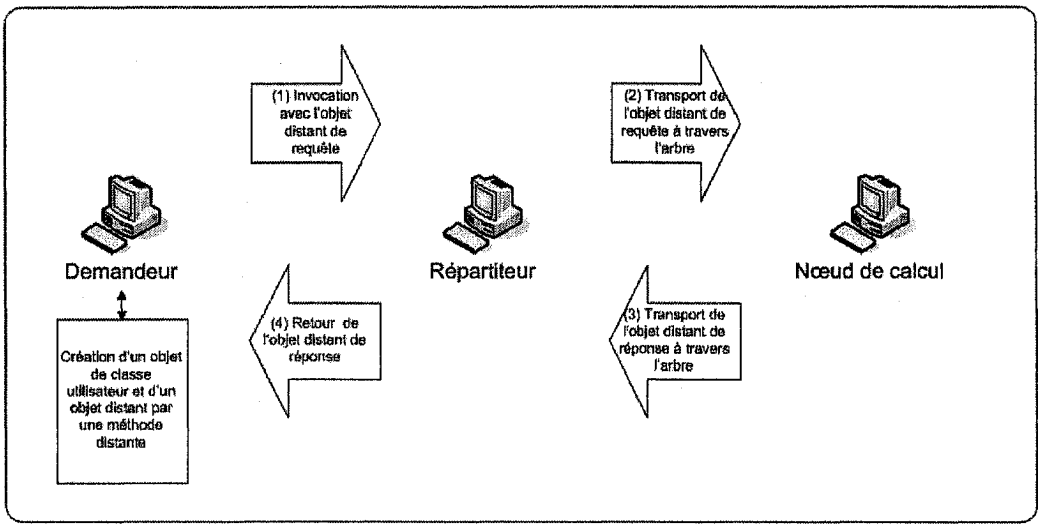

Figure 2.16 Représentation réseau de la structure d'appel distant.

La librairie de classes fournie par l'utilisateur doit contenir une méthode de classe servant d'abstraction, cette méthode doit se nommer « Invoque » avec un seul paramètre de type abstrait objet et retourner le résultat sous forme d'un objet abstrait de type objet. Cette méthode sert de point de contact abstrait afin de rendre translucide le transport dans l'arborescence. Voici la forme exacte de cette méthode de classe.

```
public static objet Invoque(objet o)
{
        ObjetDistant oDist = (ObjetDistant)o
        retourne oDist.fn(oDist.par)
}
```

Les termes de ce code sont présentés :

- public : permet l'accès à cette méthode, puisque l'appel est fait à l'extérieur de la classe
- static : fait de cette méthode, une méthode de classe. Cette particularité est importante car aucun objet de cette classe ne sera crée par le transporteur au moment de l'appel
- objet o : le paramètre est une référence à un objet « ObjetDistant »
- oDist.fn(oDist.par) : invocation de « fn » la méthode distante avec les paramètres fournis par oDist.par.

Afin de se servir de la classe « ObjetDistant », il faut ajouter une référence sur la librairie de communication de l'arborescence fournie avec les modules répartiteur/nœud. Cette librairie contient la définition et l'implantation nécessaire à l'interface utilisateur pour le transport. La dernière contrainte découle directement de l'appel interne de la méthode « Invoque », la méthode « Fn » ainsi invoquée doit recevoir et retourner un type d'objet abstrait « objet ». Les méthodes faisant partie de l'ensemble invocable doivent être accessibles « public ».

L'exemple suivant montre la méthode qui reçoit un entier en paramètre et retourne un tableau d'entiers croissant de 0 à cet entier. Le nom de la librairie de calcul est « NS.DLL »

```
EspaceDeNom NomSimple
{
        Classe Publique ClasseSimple
        {
                public static objet Invoque(objet o)
                {
                        ObjetDistant oDist = (ObjetDistant)o
                        retourne oDist.fn(oDist.par)
                }

                public objet Simple(objet o)
                {
                        // Conversion connue
                        entier n = (entier)o

                        // Création d'un tableau v de dimension n
                        tableauEntier v = nouveau tableau [n]

                        pour i de 0 à n-i
                                v[i] = i;
                        fin pour

                        // Retourne v sous forme abstraite objet
                        retourne v
                }
        }
}
```

Cet exemple est trivial et ne sert qu'à la compréhension du modèle d'appel dont voici la syntaxe au niveau de l'appel de l'interface:

```
...
// Création d'un objet de la classe utilisateur
NomSimple.ClasseSimple cs = nouveau NomSimple.ClasseSimple()

// Un indicateur pour la longueur du tableau à recevoir
entier x = 10

// Création d'un objet de méthode distante
MethodeDistante mDist = nouveau MethodeDistante()

// Affectation des champs de la méthode
mDist.nomDLL        = "... NS.DLL"
mDist.nomClasse     = "NomSimple.ClasseSimple"
mDist.fn            = nouveau ObjetDistant.FnObjetDistant(cs.Simple)
mDist.obj           = cs
mDist.par           = (objet)x

// Invocation dirigée de la méthode exécution asynchrone InvoqueUn
mDist.InvoqueUn(Nœud(i), Interface, clé)

// Attente du résultat en mode asynchrone
tantque (mDist.termine == FAUX)
        MiseEnVeille(5 ms)

// Vérification de complétude de la méthode (système non-dédié)
Si (mDist.reponses existe)
{
        // Récupération du ou des résultats, InvoqueUn ne retourne qu'un
        // résultat soit reponses[0] sous forme d'un objet distant.
        ObjetDistant oDist = (ObjetDistant)mDist.reponses[0].valeur
        // Conversion du format du résultat
        TableauEntier v = (TableauEntier)oDist.rep
        ...
}
```

L'exemple d'appel illustré ci-haut permet une versatilité entre l'appel dirigé simple et les appels multiplexés que nous montrons dans la section suivante. Puisque ces méthodes sont asynchrones, l'appel de « mDist.InvoqueUn » n'est pas bloquant et le résultat revient dans un temps non déterminé, c'est pourquoi nous devons attendre la fin du traitement dans une boucle d'attente.

Le passage en mode parallèle d'un calcul ou d'une demande peut se faire par des appels itératifs dirigés ou non dirigés et ne contenir qu'une seule boucle d'attente, l'exemple présent est très simple et ne relève peut-être pas la puissance de cette versatilité. Le répartiteur n'est sollicité qu'à l'appel de la méthode InvoqueUn.

La formulation du code présenté précédemment montre de manière complète la transparence au niveau du transporteur sur la classe qu'il transporte, il ne connaît donc pas l'existence de la classe « ClasseSimple », il ne transporte qu'un « ObjetDistant » à travers une « MethodeDistante » dont la valeur effective est un objet de la « ClasseSimple ».

2.3.1.2 Méthode distante

Nous avons vu l'utilisation d'un objet « MethodeDistante » qui n'est ni plus ni moins qu'une classe d'exécution au dessus de la classe « ObjetDistant » permettant le transport et l'exécution de méthode utilisateur distante. Il y a deux modes possibles d'appel avec variantes:

- InvoqueUn
- InvoqueTous:

Un champ d'information « termine » indique si la méthode s'est complétée.

Le champ « reponses » contient à la fin de la transaction (« termine » est « vrai »), la ou l'ensemble des réponses générées par l'appel. De manière conventionnelle, il s'agit d'un tableau de réponses qui n'en contient qu'une, si une variante de « InvoqueUn » est appelée.
Nous pouvons invoquer de manière dirigée ou non dirigée, cette flexibilité permet de recueillir des informations sur des nœuds ciblés (mode dirigé) ou de manière transparente à l'utilisateur en ne connaissant pas le point d'exécution (mode non dirigé).

Ainsi en mode dirigé, nous pouvons créer une classe permettant de recueillir par exemple le taux d'utilisation d'un appareil et garder cet historique. Mais l'utilisateur dans la majorité des cas, n'a pas à connaître le point de calcul et n'a pas à savoir où le calcul a été fait.

Cependant l'invocation doit être faite sur un répartiteur, nous devons donc connaître le PCIP d'un répartiteur dans l'arbre. L'invocation n'est pas obligatoirement fait sur la racine, mais simplement sur un répartiteur quelconque dans l'arbre qui est racine d'un sous arbre dans lequel la requête est effectué. Nous trouvons comme paramètre d'appel une clé texte, c'est la clé qui permet l'encodage des transferts entre les nœuds dans notre arbre ainsi que l'appel initial du module utilisateur.

2.3.1.3 InvoqueUn (mode dirigé)

Le titre de cette méthode s'explique par lui-même, il s'agit d'un appel distant sur un nœud ciblé (dirigé). Pour se faire, il faut cependant connaître le PCIP du nœud distant. Une méthode permet de recueillir la liste des nœuds disponibles sur un répartiteur donné et à partir de cette liste, un choix peut être fait.

L'appel comprend trois paramètres :

- PCIP répartiteur
- PCIP nœud visé
- Clé texte d'encodage pour l'arbre

2.3.1.4 InvoqueUn (mode non dirigé)

L'appel de cette méthode force l'exécution de « Fn » sur un nœud quelconque disponible dans le sous arbre de ce répartiteur.

L'appel comprend deux paramètres :

- PCIP répartiteur
- Clé texte d'encodage pour l'arbre

2.3.1.5 InvoqueUn (mode non dirigé avec multiplicité)

L'appel de cette méthode force l'exécution de « Fn » sur une multiplicité N de nœuds quelconques disponibles dans le sous arbre de ce répartiteur.

L'appel comprend trois paramètres :

- PCIP répartiteur
- Clé texte d'encodage pour l'arbre
- La multiplicité, le nombre de nœuds qui devront exécuter « Fn »

Nous pouvons donc conclure que InvoqueUn sans multiplicité est une variante avec une multiplicité de un.

2.3.1.6 InvoqueTous (mode non dirigé)

L'appel de cette méthode force l'exécution de « Fn » sur tous les nœuds disponibles du sous arbre de racine répartiteur à qui la requête est faite.

L'appel comprend deux paramètres :

- PCIP répartiteur
- Clé texte d'encodage pour l'arbre

2.3.1.7 Distribution des librairies de calcul

Afin d'augmenter la flexibilité de notre système, une fonctionnalité de transport de librairies de calcul est implantée qui permet à l'usager de distribuer ses propres librairies sur tous les nœuds d'un sous arbre d'un répartiteur interface. Puisque le système ne connaît pas ce qu'il transporte, l'ajout d'une librairie dans l'arborescence consiste à ce que chaque nœud puisse appeler une fonction native d'une librairie utilisateur, donc chaque nœud doit avoir cette librairie. Cette fonctionnalité permet de distribuer ces librairies à travers les nœuds et ainsi rendre disponible les fonctions natives de l'utilisateur.

2.3.2 Requête non native

Cette fonctionnalité permet d'interroger des modules qui ne sont pas compatibles en terme de langage de programmation mais dotés de fonctionnalités réseaux. L'interface tout comme dans le mode natif reçoit des requêtes par un « socket TCP» mais cette fois, le format exact de la requête n'est pas connu. Nous devons établir à l'encontre de l'abstraction de transport, une forme prédéterminée pour le transport sous la forme d'une entête de transport qui sert d'interface pour la traduction de l'information entre l'appelant et l'appelé. Cette forme d'échange demande la reconstruction du module d'interface et par le fait même, la reconstruction des différents nœuds et répartiteurs de notre arborescence.

La raison de l'existence de ce mode est de permettre à des applications tierces de se servir du transport pour résoudre des problèmes dont la solution séquentielle existe déjà et dont la forme distribuée est décomposable dans ce module tiers. Il s'agit d'adapter l'arborescence de répartition à un modèle concret existant à la solution distribuée tierce. Les requêtes tierces « RT » sont acheminées vers l'interface de l'arborescence qui les traduit en mode natif pour le transport. Cependant une classe de conversion/transport doit être élaborée à l'intérieur même de l'arborescence et ainsi demande la reconstruction unique pour l'adaptation à ce modèle tiers.

Une fois les normes d'interface entre le modèle tiers et l'arborescence établies, l'interface fait la conversion et le transport est possible de manière abstraite (Figure 2.17).

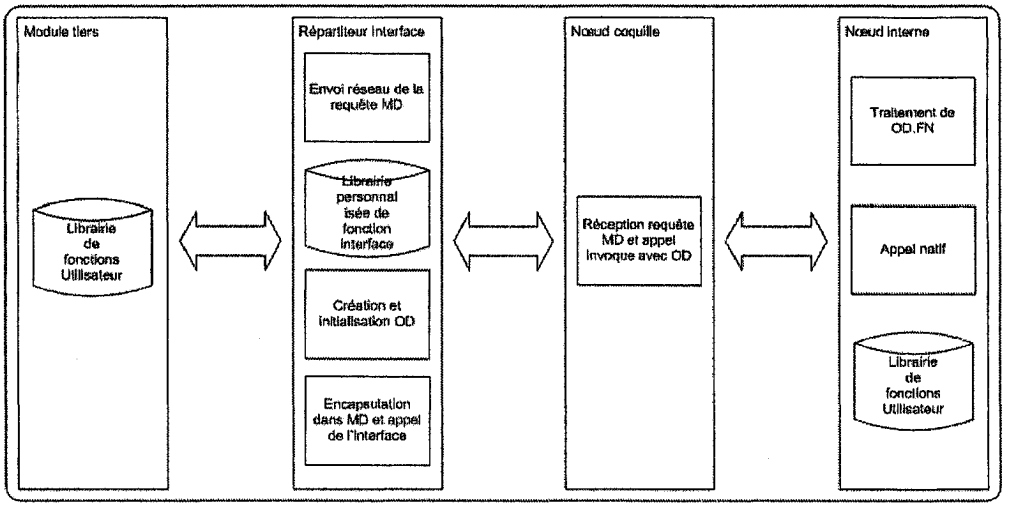

Figure 2.17 Encapsulation vs modèle tiers.

La réponse constitue un autre problème, car l'encapsulation est effectué par l'interface à l'appel, nous avons donc deux choix pour le retour : soit du nœud vers l'appelant par l'arborescence ou soit directement du nœud vers l'appelant. Dans une arborescence de profondeur élevée, le temps de transfert entre un appelant et une feuille profonde (nœud de calcul) de notre arbre se répète à chaque niveau de l'arbre. Puisque nous avons minimisée cette information, l'incidence est relativement réduite. De manière générale les réponses sont largement supérieures en tailles aux requêtes de calcul, ce qui augmente drastiquement le temps de réponse en fonction de la profondeur du nœud. Le cheminement par l'arborescence de réponses de taille élevée n'est pas souhaitable et doit être contourné par une réponse directe du nœud vers l'appelant. Si nous désirons acheminer la réponse par l'arborescence en mode non natif, une encapsulation supplémentaire doit être incorporée dans le nœud interne.

2.4 Performances des nœuds en fonction de la demande

Lorsqu'un nœud de calcul s'inscrit à un arbre, il doit fournir sa valeur qualitative, celle-ci est constituée de la vitesse du micro processeur et la capacité en mémoire vive.
Lors de la prise de décision pour la distribution d'une requête dans notre système, nous devons prendre en considération cette performance afin de tenter de minimiser le temps global de calcul dans notre arbre pour les appels non dirigés. Cette information est minimale et fait partie de chaque envoi dans les paquets d'information puisque la présence en disponibilité d'un nœud est relative à sa mise sous tension, ce qui peut varier tout au long du cycle d'activité de l'arborescence distribuée. En minimisant ces valeurs, nous pouvons inclure celle-ci dans les transferts périodiques (1/f) sans pour autant engorger le réseau.

Le système distribué de calcul ne connaissant pas ce qu'il transporte, ne sait pas la charge implicite d'un calcul demandé par l'utilisateur. Il dirige les calculs dans l'ordre de disponibilité des nœuds qui sont triés par leurs valeurs qualitatives en ordre décroissant. Ce tri doit être maintenu tout au long du cycle d'activité de l'arborescence et puisque qu'une demande peut être faite sur n'importe quel répartiteur connu de l'arbre, chaque répartiteur doit maintenir une liste pour les nœuds qui dépendent de lui dans le sous arbre dont il est racine.

2.5 Pile de distribution des calculs

Lors de la demande de calculs par requêtes natives ou non natives, les demandes sont emmagasinées dans une pile de traitement. À la disponibilité d'un nœud pour calcul, cette pile est consultée, si elle contient des demandes en suspens elles sont séquentiellement acheminées vers les nœuds libres en tenant compte de la multiplicité. Pour une requête de multiplicité élevée N, elle peut bloquer le système pendant un temps de N*E (l'espérance de disponibilité d'un nœud), la définition propre relative aux équipements concernés qui fait que E tend vers zéro et que N*E devrait être minime voire nul.

Les requêtes sont exécutées de manières itérative et séquentielle au fur et à mesure de la disponibilité en nœuds pour leur exécution (Figure 2.18).

Dans le paquet informatif envoyé par les nœuds, un indicateur de traitement doit être fourni pour indiquer en plus de la disponibilité, l'état de traitement de requête. Ce qui dans le mode natif n'est pas un problème puisque l'exécution des méthodes est synchrone, c'est à dire la fin de l'exécution est attendue avant de continuer. Pour le mode non natif, le phénomène est plus complexe puisque les fonctions non natives de l'interface peuvent répondre directement à l'appelant, dans cette situation nous devons faire des appels asynchrones de demande au module non natif et celui-ci doit prendre en charge la transmission de sa fin de calcul au répartiteur appelant. Il existe une variante d'appel non natif synchrone qui permet de contourner le problème par la mise en attente de la complétude du calcul et donc la connaissance du nœud coquille sur la fin du calcul non natif. Cependant, l'exécution ne peut pas être interrompue de manière logicielle par le nœud dans le cas ou un usager utilise l'ordinateur. Lors de l'implantation d'une personnalisation par module tiers, le système tend de plus en plus vers un modèle dédié et que cette incidence n'a que peu d'impact si les calculs sont de faibles durées.

Le concept d'identification de terminaison de calcul est plus complexe qu'il ne le semble car le nœud coquille en appel asynchrone ne répond pas directement à son maître et ainsi récursivement jusqu'à l'appelant, c'est pourquoi le nœud interne dans sa partie non native doit répondre de la disponibilité du nœud au maître de celui-ci qui transmet par la suite l'information par récursivité jusqu'à l'appelant. Dans la figure 2.19, un appel non natif du demandeur par R1 et dirigé vers N8 doit indiquer une indisponibilité de N8 au niveau de R3 et R1. Cette indisponibilité doit être signalée par la coquille de N8 qui est responsable de la transmission du paquet informatif qui ne connaît pas cependant l'état d'exécution en mode d'appel asynchrone. C'est pourquoi, nous utilisons le mode synchrone avec retour direct entre N8 et le demandeur pour minimiser les transferts et ainsi la coquille de N8 connaît l'état d'exécution de la fonction non native par son attente.

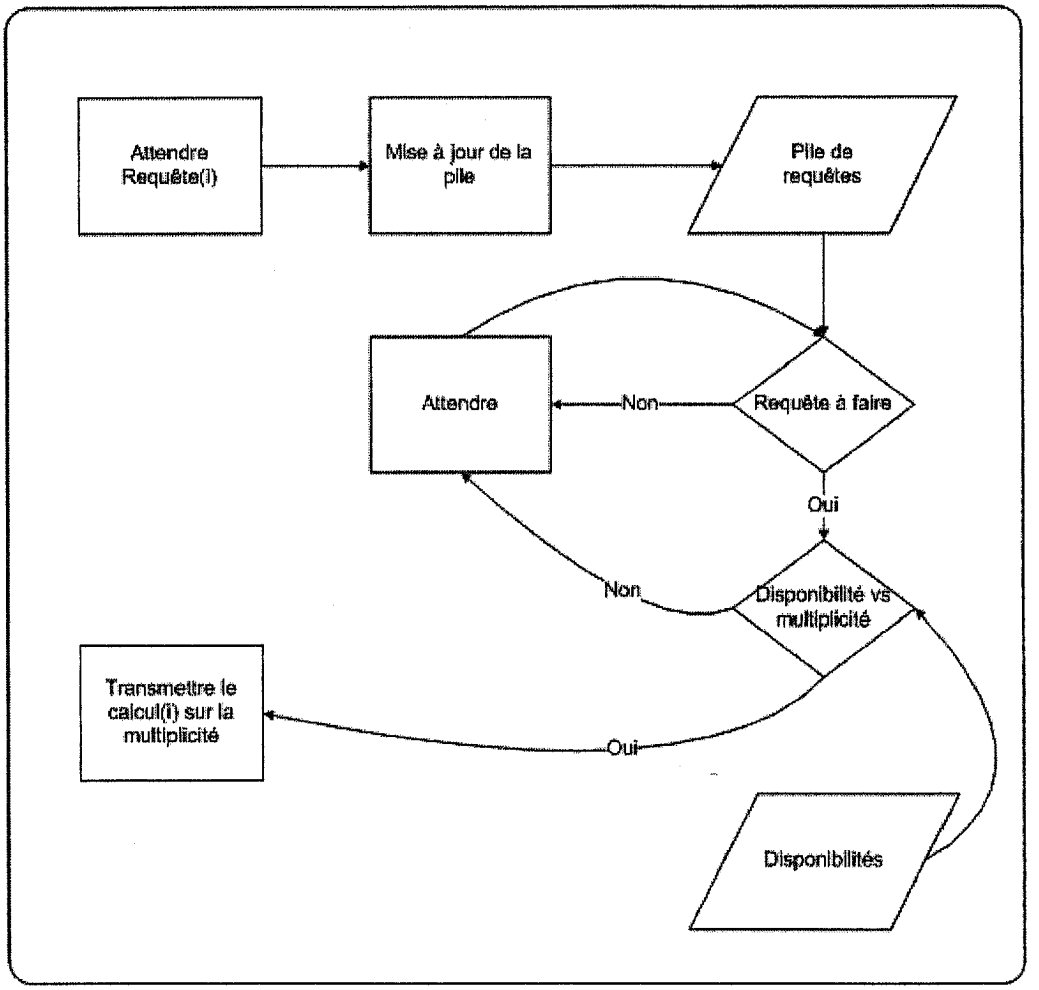

Figure 2.18 Pile d'exécution des requêtes.

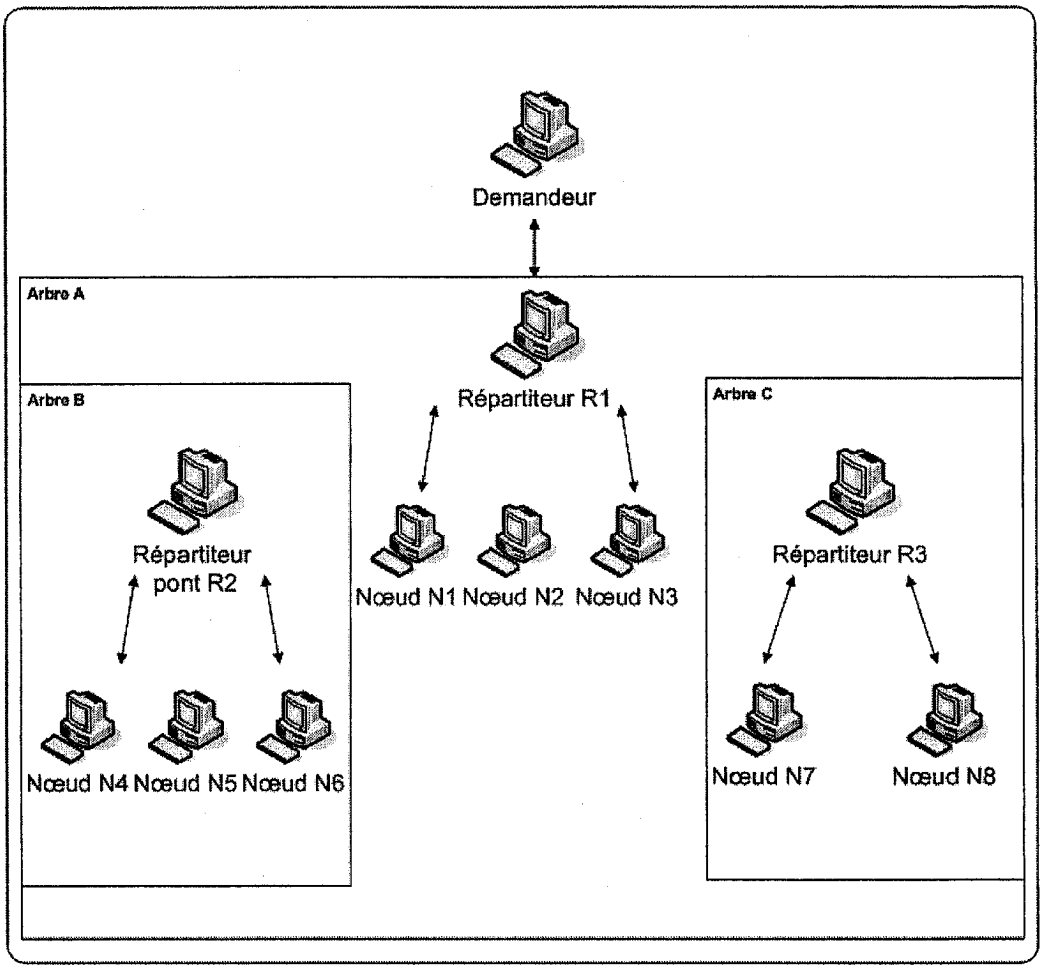

Figure 2.19 Arborescence et requête non native.

CHAPITRE 3

RÉSULTATS

3.0 Environnement de test

Pour faire une batterie de tests complète sur un modèle d'architectures distribuées, un nombre considérable d'appareils est nécessaire. Dans le cadre de cette recherche, les équipements ne sont pas disponibles en grand nombre et nous devons concentrer ceux-ci sur un sous-ensemble dans le simple but de prouver la viabilité du concept. Les résultats escomptés doivent faire foi de l'objectif et être extrapolés sur un ensemble d'envergure qui ne nous est pas disponible.

Notre sujet possède deux volets soit : la mise en parallèle de la phase du maillage de modèles et la construction d'une architecture distribuée en regard des équipements disponibles. Les critères d'évaluation sont : Dg, la grosseur des mailles et Eps, l'écart maximale des mailles avec la géométrie [François, 1998].

3.1 Mise en parallèle de la phase maillage

Il s'agit d'établir les sections du maillage qui peuvent être misent en parallèle et de concevoir les pré et post-conditions nécessaires. Il existe une phase itérative de maillage au niveau des surfaces qui permet la distribution du maillage de chacune des faces sur plusieurs nœuds de notre arborescence. Chaque maillage de surface devient un calcul dur un nœud. Les nœuds sont utilisés en file circulaire selon leur disponibilité. Les pré-conditions se présentent sous la forme d'un pré maillage qui consiste à construire le modèle 3D à partir d'un fichier dessin provenant d'un logiciel de DAO. Les post-conditions représentent la reconstruction ou l'assemblage des résultats de maillage obtenus des nœuds de calcul. Cette partie a été possible grâce à la direction et collaboration de M. Vincent François par son expertise dans le domaine du maillage.

3.2 Construction de l'architecture distribuée

Le sujet de cet œuvre couvre l'établissement d'un modèle distribué capable d'effectuer des tâches indépendantes de manière à ce que le demandeur n'ait pas à se soucier de la construction de l'arborescence et simplement demander un travail ou calcul à une interface de programmation. Les limitations d'équipements nous force à travailler sur de petits ensembles et ainsi à ne pas exploiter le caractère de multi-calcul de notre système. Nous avons donc utilisé une architecture simple qui ne comporte qu'un niveau de profondeur et un nombre limité d'appareils qui sans être totalement dédiés, ne font que participer à notre essai sans faire rien d'autre d'importance.

3.3 Méthode de comparaison et critères d'optimisation

Le temps de calcul global obtenu est comparé entre le modèle séquentiel et distribué afin d'obtenir un barème de comparaison de l'optimisation entre les modes séquentiel et distribué. Dans chacun de nos résultats, nous présentons les résultats séquentiels en comparaison aux résultats obtenus dans le mode distribué en raffinant le maillage afin d'augmenter le temps de maillage et ainsi comparer l'effet d'optimisation par le modèle distribué.

3.4 Essais

3.4.1 Premier essai

Nous avons utilisé un modèle de maillage simple (Figure 3.1) sur un ensemble d'appareil de différentes performances (Figure 3.2). Le modèle possède dix faces et en ne tenant pas compte de la complexité différente entre les faces nous obtenons que 2 des nœuds ont 2.5 faces à mailler par nœud, ce qui dans un monde concret nous indique que sommairement 2 nœuds maillent 3 faces et les deux autres nœuds maillent 2 faces. Nous disposions de 4 appareils que nous identifions par leur performance en rapidité de traitement : N1 = 2,8GHz, N2 = 2,6GHz, N3 = 1,8GHz et N4 1,7GHz. Les valeurs de temps pour le mode séquentiel ont été obtenues avec le plus puissant des nœuds, soit N1.

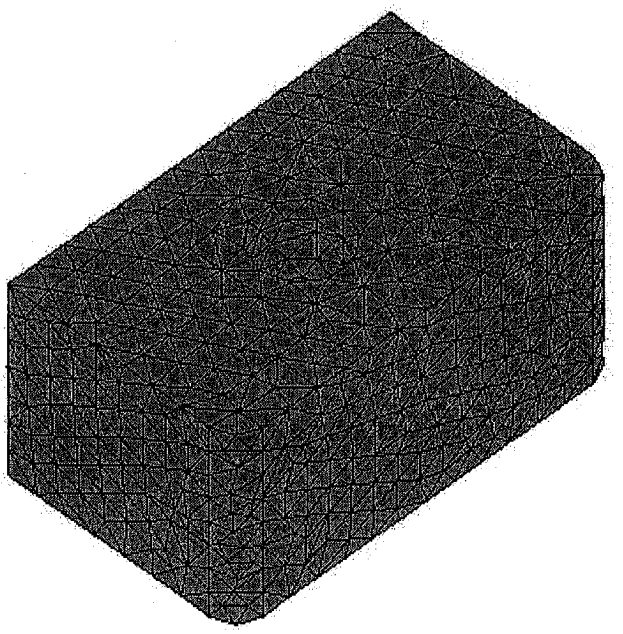

Figure 3.1 Modèle simple, cube troué (Eps=1, Dg=10)

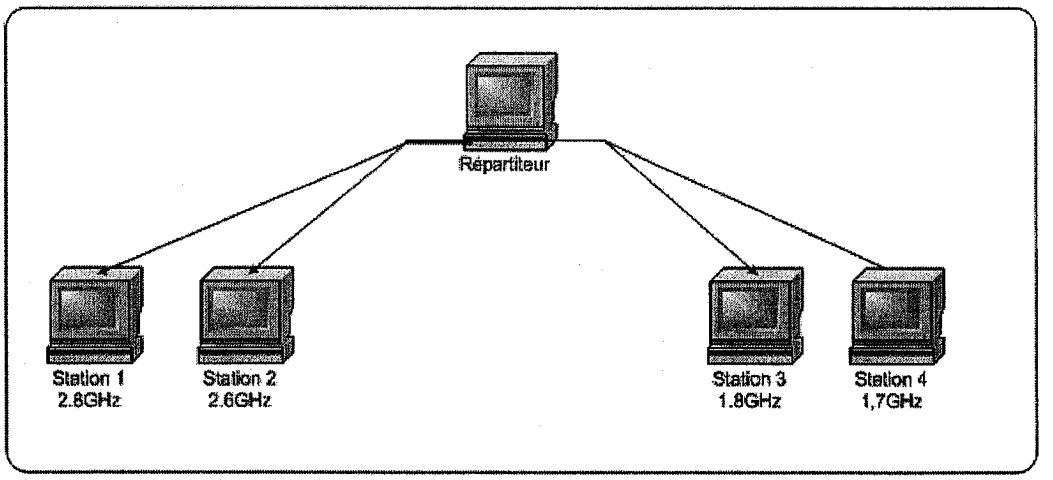

Figure 3.2 Arborescence.

Les résultats sont présentés dans le tableau suivant et ont été obtenus par la moyenne sur 3 essais. Nous notons que dans un cas, nous avons forcé le tri des faces et des nœuds afin de démontrer que dans un système dont les nœuds ne sont pas homogènes en terme de puissance de traitement, les résultats en temps peuvent varier selon la distribution.

En effet, dans un modèle à mailler, chacune des faces possède un niveau de complexité différent pour le maillage, il est donc possible et probable que les temps obtenus puissent varier et qu'une simple permutation de nœuds dans l'arborescence, influence le résultat final. Cependant, il n'existe pas de méthode d'évaluer la complexité de maillage avant le maillage sans faire un traitement qui serait aussi coûteux que le maillage lui-même, donc nous devons conclure que le résultat en temps peut varier entre un niveau minimal et optimal et que le plus faible des deux dans un modèle à plus d'un nœud reste largement mieux que le mode séquentiel.

Tableau XII

Résultats (modèle simple)

Temps Séquentiel	Temps Distribué	Nombre de noeuds	Eps	Dg
4,95	5,25	4	1	10
113	40	4	1	2,5
154	86	4	1	2,0
154	82	3 (trié)	1	2,0
154	107	2	1	2,0
154	196	1	1	2,0

Nous constatons que sur un problème simple (Eps=1 Dg=10) nous avons une pénalité de 0,3 sec ce qui représente 0,3/4,95=0,06 (6%) de dégradation, ce qui est peu. Cette dégradation est liée au temps de décomposition et reconstitution du modèle en mode distribué. Ce temps s'améliore radicalement avec la complexité du maillage (Eps=1 Dg=2) (Figure 3.3) pour donner un gain 86/154=0,558 (56%) . La courbe respective de performance (figure 3.3.1) donne une vue d'ensemble du processus distribué qui gagne rapidement en gain sur le maillage séquentiel par interpolation. Nous pouvons nous permettre d'extrapoler le tout afin de déterminer un gain croissant en fonction de la complexité du maillage et du nombre de faces. Notons que le mode distribué à un nœud n'implique qu'une dégradation de 21,4% (1 – (154/196)), ce qui nous permet de ne fonctionner qu'en mode distribué car il sera peu probable de n'avoir qu'un nœud dans notre arborescence.

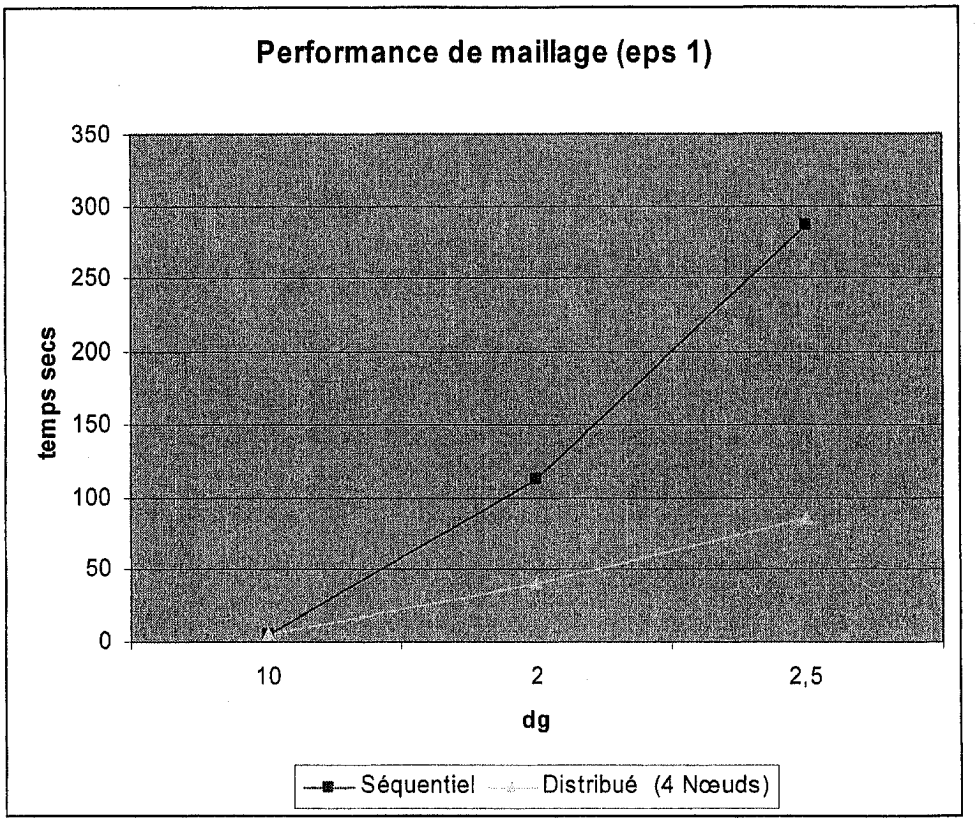

Figure 3.2.1 Performances

3.4.2 Deuxième essai

Lors du premier essai, nous avons constaté que le gain de temps obtenu était directement lié à la proportion entre le nombre de nœuds de calcul et le nombre de faces à mailler en respect avec la complexité individuelle des surfaces. C'est pourquoi le deuxième essai a été réalisé avec 10 nœuds de calcul sur une architecture similaire à un niveau mais avec différentes valeurs de Dg et Eps et avec différents modèles.

3.4.2.1 Cube troué

Dans cet essai, le modèle à mailler est un cube troué (Figure 3.3) comportant 10 faces.

C'est pourquoi le temps total est égal au maillage de la plus complexe face additionné du temps d'ensemble : pré maillage, distribution, retour du maillage et assemblage des résultats. Le temps d'ensemble pour un modèle donné est presque constant en regard au trafic du segment de réseau sur lequel est effectué l'essai. Les résultats présentés dans le tableau ci-bas indiquent un gain léger même sur une complexité faible. Ce gain croît rapidement avec la complexité non linéaire d'Eps et de Dg. Plusieurs des résultats sont obtenus rapidement et les faces plus complexes terminent le processus dû au temps de calcul plus long, ce phénomène confirme la relation discutée précédemment à l'effet que la distribution et l'efficacité du réseau sont presque constantes.

Tableau XIII

Résultats (cube troué)

Temps Séquentiel	Temps Distribué	Nombre de noeuds	Eps	Dg
3,14	2,5	10	1	10
45,89	12,4	10	0,4	3

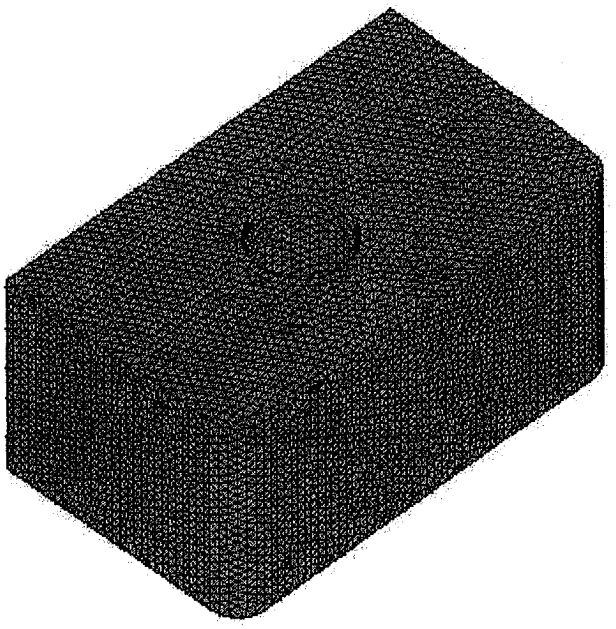

Figure 3.3 Modèle simple, cube troué (Eps=1 Dg=3).

3.4.2.2 Support d'arbre d'entraînement

Dans cet essai, le modèle à mailler est un support d'arbre d'entraînement (Figures 3.4 et 3.5) comportant 25 faces.

Tableau XIV

Résultats (support d'arbre)

Temps Séquentiel	Temps Distribué	Nombre de nœuds	Eps	Dg
29,8	11,3	10	0,025	0,2
251,7	102	10	0,01	0,1

Dans l'essai Eps = 0,01 et Dg = 0,1, nous avons noté que le maillage de 23 des 25 faces était obtenu en 27 secondes et que les faces résiduelles retardaient la terminaison du processus. Ne pouvant pas avoir de facteur préliminaire d'évaluation de la complexité avant le maillage, nous notons une relation de distribution qui influence le temps global et qu'une architecture possédant un nombre de nœuds supérieur ou égal au nombre de faces serait optimale.

Figure 3.4 Arbre entraînement (Eps=0,025 Dg=0,2).

Figure 3.5 Arbre entraînement (Eps=0,01 Dg=0,1).

3.4.2.2 Éjecteur

Dans cet essai, le modèle à mailler est un éjecteur (Figures 3.6 et 3.7) comportant 31 faces.

Tableau XV

Résultats (éjecteur)

Temps Séquentiel	Temps Distribué	Nombre de nœuds	Eps	Dg
17,53	8	10	0,75	7
178,79	51	10	0,4	3

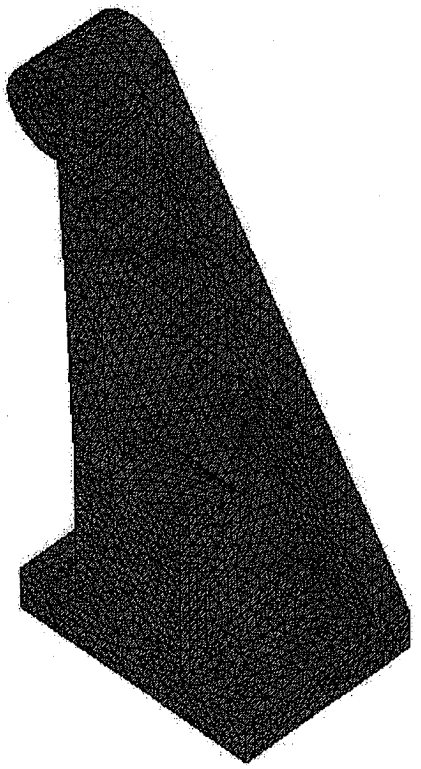

Figure 3.6 Éjecteur (Eps=0,75 Dg=7).

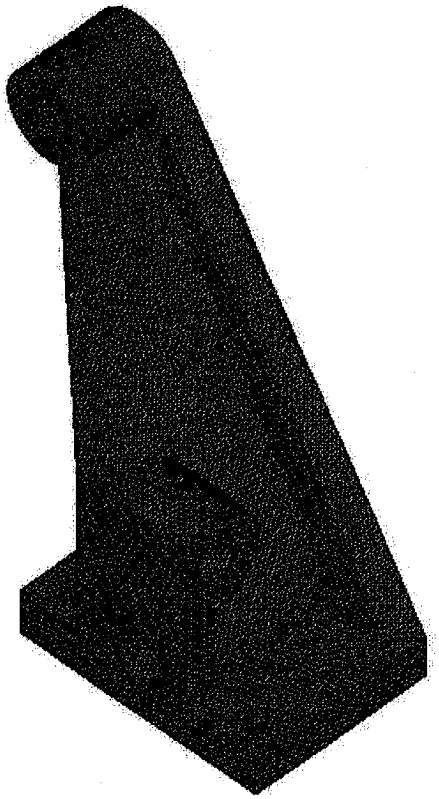

Figure 3.7 Éjecteur (Eps=0,4 Dg=3).

DISCUSSION ET INTERPRÉTATION DES RÉSULTATS

Nous avons observé dans chacun des essais que le modèle distribué, même sur des surfaces à mailler de faible complexité n'implique qu'un léger ralentissement du processus total de maillage et qu'il était rapidement bénéfique d'utiliser la distribution. Les ralentissements implicites à la décomposition, au transfert et à l'assemblage des résultats se voient rapidement estompés lorsque la complexité d'eps et dg se raffine.

C'est pourquoi, il est possible de croire que si le nombre de nœuds est supérieur ou égal au nombre de faces à mailler, le résultat en temps serait optimal. Cependant si la performance des nœuds n'est pas semblable, le temps total est affecté par la distribution des faces sur les nœuds et il n'est pas possible de déterminer où doit être effectué le maillage des faces les plus complexes puisque nous ne connaissons pas la complexité avant de les avoir maillés. La distribution par rapport à la performance des nœuds se résume donc à une distribution séquentielle rotative sur les nœuds disponibles.

Il ne nous a pas été permis de bénéficier d'assez de nœuds pour créer une architecture distribuée à plusieurs niveaux afin de tester l'implication des répétitions aux sous répartiteurs d'un tel arbre. De plus, les nœuds utilisés n'étaient pas occupés au moment des calculs et rien ne nous a permis de pouvoir utiliser les multiples distributions de calcul de même face dans le but de voir le comportement lors de l'arrêt prématuré d'un calcul. Bien que ces essais soient limités, la conclusion est claire tant au gain en temps de la distribution d'un maillage 2D sur une architecture distribuée. Le concept est donc viable dans ce contexte.

CONCLUSION

Plusieurs sujets furent laissés en plan, dû majoritairement au manque d'équipement, mais les résultats obtenus confirment la viabilité de l'application du modèle d'architecture distribuée sur le traitement de maillage 2D. Ces résultats nous indiquent clairement que cette expérimentation est un succès et que plusieurs facteurs permettent d'améliorer le rendement de l'application. Il faut appliquer cette approche sur un large ensemble de nœuds afin d'en démontrer l'efficacité sans extrapolation. Malgré tout, les résultats découlant d'une architecture constituée d'un grand nombre de nœuds sont pour le moment extrapolés avec un très bon degré de certitude et ce sur la base des résultats actuels.

Ce mode de calcul possède un potentiel très fort au sein de la recherche effectuée à l'UQTR sur l'intégration de la méthode des éléments finis dans le processus de CAO/FAO. En effet le développement de cette recherche aboutit à la construction d'algorithmes de plus en plus lourds qui agissent sur des modèles de plus en plus complexes. La performance de l'ensemble de ces algorithmes peut être augmentée de manière notoire en utilisant le calcul distribué sans avoir à paralléliser le code existant.

RECOMMANDATION(S)

L'arborescence présentée dans ce document répond aux critères que nous avons fixés dans le modèle proposé. L'impact en temps de réponse du réseau s'est avéré être moins considérable qu'anticipé, en majorité dû à la modernisation du réseau institutionnel, de sa fiabilité et de sa stabilité.

Le modèle exploité lors des tests ne comprend que des appareils dédiés et la phase de traitement des libérations de nœuds avant la fin de calculs et la multiplicité ne fut pas explorée. Tout ce secteur mérite un plus grand examen et doit être une suite logique à cet œuvre, par laquelle nous pourrons exploiter la puissance non utilisée des postes de travail qui sont nombreux et si peu utilisés.

Bien que globalement le sujet a prouvé sa viabilité, l'interface entre le maillage et l'arborescence était lourd due essentiellement à la différence des compilateurs utilisés. Plusieurs « workaround » furent utilisés pour contrecarrer la complexité d'incompatibilité entre ces deux mondes. Malheureusement, nous prouvons la faiblesse d'incompatibilité qui existe encore de nos jours et qui limite souvent les résultats sans pourtant faire partie du problème théorique. C'est pourquoi, le sujet doit être exploré en rendant compatible à la source ces deux mondes qui sont : le maillage et les répartiteurs. La mise en œuvre des tests a été laborieuse, du fait ce cette incompatibilité qui força la distribution manuelle des modèles à mailler ainsi que du pré maillage, l'intégration harmonieuse est souhaitable et permet d'accroître les performances entre le modèle séquentiel et distribué.

Mon travail se termine ici et je souhaite que quelqu'un regarde cet essai et y voit un potentiel ou LE potentiel dormant et qu'il puisse en tiré profit.

BIBLIOGRAPHIE

ATTIYA, H.,AND WELCH, J.
Distributed Computing – Fundamentals, Simulations and Advanced Topics.
McGraw-Hill (1998)

BATOZ J. L., DHATT G.
Modélisation des structures par éléments finis.
Edition Hermès, 3 tomes, 1990.

BERSHAD, B., ZEKAUSKAS, M. AND SAWDON, W.
The Midway distributed shared memory system.
In Proceedings IEEE COMPCON Conference, IEEE, pp.528-37 (1993)

BLAIR, G.S. AND STEFANI, J-B.
Open Distributed Processing and Multimedia.
Harlow, England: Addison-Wesley (1997)

CHERITON, D. AND MANN, T.
Decentralizing a global naming service for improved performance and fault tolerance.
ACM Transactions Computer Systems, Vol.7, No.2, pp. 147-83 (1989)

DHATT G., TOUZOT G.
Une présentation des éléments finis.
Edition Maloine, 543 p (1984)

EL ABBADI, A., SKEEN, D. AND CRISTIAN, C.
An efficient fault-tolerant protocol for replicated data management.
In 4th Annual ACM SIGACT/SIGMOD Symposium on Principles of Data Base Systems, Portland Ore. (1985)

FIDGE, C.
Logical Time in distributed Computing Systems.
IEEE Computer, Vol.24, No. 8, pp. 28-33 (1991)

FRANÇOIS, V
Méthodes de maillage et de remaillage automatiques appliquées à la modification de modèle dans le contexte de l'ingénierie simultanée
Thèse, UNIVERSITÉ HENRI POINCARÉ, NANCY 1 (1998)

GOKHALE, A. AND SCHMIDT, D.
Measuring the Performance of Communication Middleware on High-Speed Networks.
Proceeding of SIGCOMM '96, ACM, pp.306-17 (1996)

GUTTMAN, E.
Service Loocation Protocol: Automatic Discovery of IP Network Services.
IEEE Internet Computing, Vol. 3, No.4, pp. 71-80 (1999)

KLEINROCKS, L.
Nomadicity: anytime, anywhere in a disconnected world.
Mobile Networks and Applications, Vol.4, pp. 351-7. (1997)

KONSTANTAS, D., ORLAREY, Y., GIBBS, S. AND CARBONEL, O.
Distributed Music Rehearsal.
In Proceedings International Computer Music Conference 97.
http://cuiwww.unige.ch/OSG/publications/TR97/TR97Contents.html (1997)

MOSS, E.
Nested Transactions, An Approach to Reliable Distributed Computing.
MIT Press. (1985)

NAGLE, J.
Congestion Control in TCP/IP
Internetworks, Computer Communications Review, Vol.14, pp.11-17, October. (1984)

OBJECT MANAGEMENT GROUP.
Index to CORBA services.
OMG: Framingham, Mass. http://www.omg.org/corba/library/csindx.html

REDMOND, F.E.
DCOM: Microsoft Distributed Component Model.
IDG Books Worldwide (1997)

SANDHU, R. COYNE, E., FELSTEIN, H. AND YOUMAN, C.
Role-Based Access Control Models.
IEEE Computer, Vol. 29,No.2, February (1996)
http://www.list.gmu.edu/journal.htm

SEETHARAMAN, K. (ed)
Special Issues: The CORBA Connection, Comms.
ACM, October, Vol.41, No. 10. (1998)

SHMIDT, D.
Evaluating architectures for multithreaded objet request brokers, Comms.
ACM, Vol.44, No. 10, pp. 54-60. (1998)

STELLING,P.,FOSTER,I.,KESSELMAN,C.,LEE,C.ANDVON LASZEWSKI,G.
A Fault Detection Service for Wide Area Distributed Computations,
Proceedings 7th IEEE Symposium on High Performance Distributed Computing,
pp. 268-78. (1998)

SUN MICROSYSTEMS
Java Remote Method Invocation
http://java.sun.com/products/jdk/1.3/docs/guide/rmi/spec/rmiTOC.html

VAN STEEN, M., HAUCK, F., HOMBURG,P. AND TANENBAUM, A.
Locating objects in wide-area systems.
IEEE Communication, Vol.36, No. 1, pp.104-109. (1998)

WEIKUM, G.
Principles and Realization Strategies of Multilevel Transaction Management.
ACM Transactions Database Systems, Vol.16, No.1 pp.132-40 (1991)

WEISMANN, M., PEDONE, F., SCHIPER, A., KEMME, B. AND ALONSO, G.
Understanding replication in databases and distributed systems.
In Proceedings 20th International Conference on Distributed Computing Systems (ICDCS'2000).
Tapei, Republic of China, IEEE. (2000)

ZIENKIEWICZ O.C.
La méthode des éléments finis
Edition Ediscience. (1977)