UNIVERSITE BADJI MOKHTAR-ANNABA
FACULTE DES SCIENCES DE L'INGENIORAT
DEPARTEMENT D'INFORMATIQUE

جامعة باجي مختار – عنابـــــــــــة
كلية علـــــــــوم الهندســـــــة
قسم الإعـــــــلام الألـــــــــي

SYSTEMES DISTRIBUES ET SERVICES WEB

Programme de Première Année Master en Informatique
Option Ingeniérie des Logiciels Complexes

SUPPORT DE COURS REALISE PAR
PR DJAMEL MESLATI

VERSION 2016

CHAPITRE 2

ARCHITECTURE DES SYSTEMES DISTRIBUES

CONTENU

2.1 TAXONOMIE DES SYSTEMES DISTRIBUES

On vise par la taxonomie le partage des différents systèmes existants en catégories ou classes de telle sorte à faciliter leur compréhension et à parvenir aux concepts fondamentaux communs à chaque classe. La recherche d'une taxonomie n'est pas une tâche facile car les systèmes distribués sont complexes et variés. Nous proposons dans ce chapitre une taxonomie à trois niveaux qui reflète la structuration en couche présentée précédemment (figure 1.5). On répartie, dans le premier niveau, les systèmes distribués en considérant leur caractéristiques matérielles (couche matérielle). Dans le second niveau, les systèmes distribués sont vus sous l'angle des systèmes d'exploitation qui les supportent. Quant au troisième niveau, il est question de processus d'application et d'architecture. Il s'agit alors d'étudier les diverses approches de structuration (architecture) des applications distribuées en termes de composants (ou processus) et répartition des rôles.

2.2 TAXONOMIE AU NIVEAU MATERIEL

D'un point de vu abstrait, un ordinateur, se compose de deux types d'entités : les mémoires et les processeurs. On peut envisager un système distribué physique comme une collection de mémoires et de processeurs interconnectés de telle sorte à pouvoir communiquer. L'interconnexion peut être faite de diverses façons en utilisant des

technologies variées ce qui donne lieu au schéma de taxonomie de la figure 2.1. Quelque soit le système distribué considéré, il est possible de le classer dans l'une des quatre catégories de la figure 2.1.

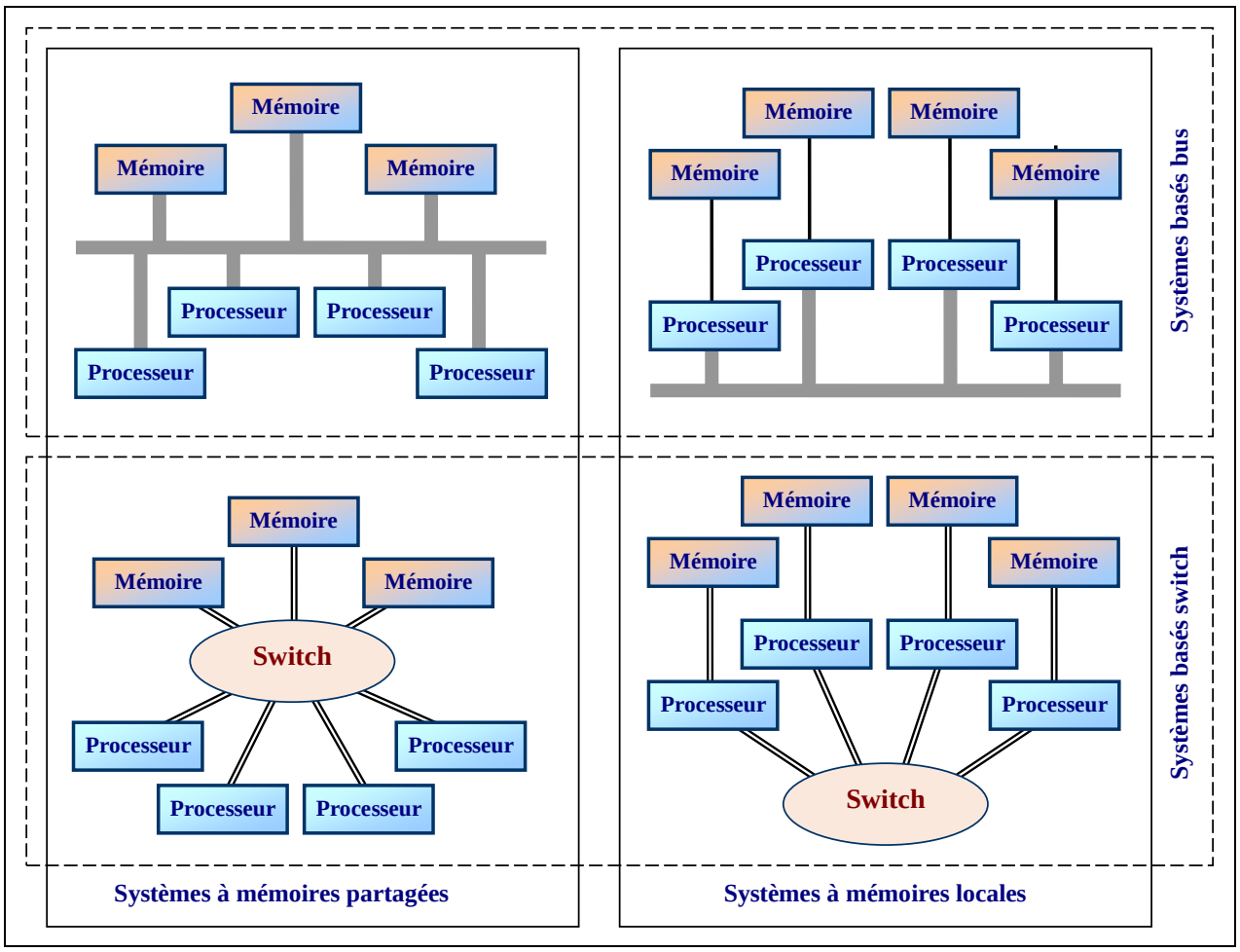

Figure 2.1. Interconnexion des processeurs et des modules de mémoire

Les systèmes à mémoires partagées sont souvent appelés *systèmes multiprocesseurs*. Les systèmes à mémoires locales, dits *systèmes multiordinateurs*, ne partagent pas de mémoires communes mais chaque processeur dispose de sa propre mémoire dont il est le seul à pouvoir y accéder.

Dans les *systèmes multiprocesseurs basés bus*, les processeurs est les modules de mémoire (un ou plusieurs) sont connectés à un bus commun de telle sorte que tous les modules de mémoire soient accessibles à n'importe quel processeur. Dans ce type de configuration, le bus constitue un goulot d'étranglement qui baisse considérablement les performances. Ce problème peut être réduit par l'utilisation de mémoires caches locales à chaque processeur. Mais, à leur tour, les caches doivent être maintenus cohérents, ce qui n'est pas une tâche facile.

Dans les *systèmes multiprocesseurs basés switch*, les processeurs et les modules de mémoire sont reliés par un dispositif de communication (réseau) utilisant des switchs (réseau de commutateurs). Lorsque le commutateur entre un processeur

Pr1 et un module mémoire M1 est ouvert, Pr peut accéder à M1. Le nombre de switch peut être important engendrant un coût prohibitif. Pour réduire ce coût, il est possible d'envisager des réseaux avec des configurations diverses. Voir figure 2.2.

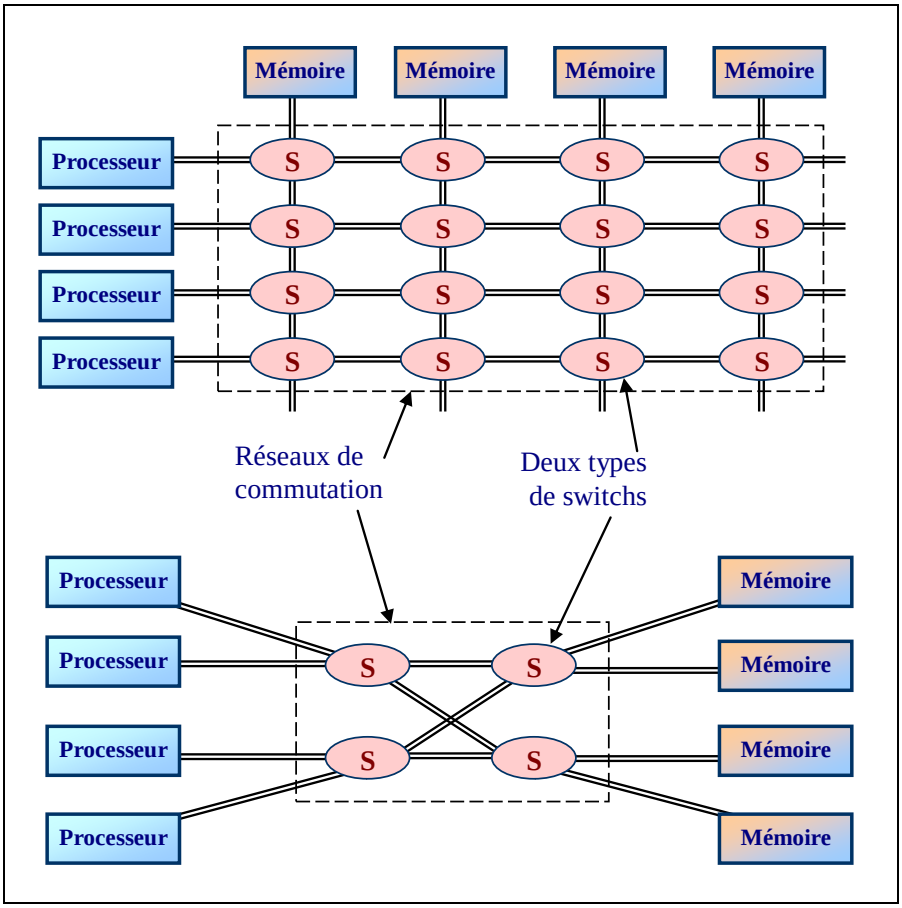

Figure 2.2. Exemples de réseau de commutation

La construction des systèmes à mémoires locales (multiordinateurs) est plus simple que celle des systèmes multiprocesseurs. Les *systèmes à mémoires locales basés bus* sont construits à partir d'ordinateurs (processeur + mémoire) identiques (on les qualifie de systèmes homogènes). Les processeurs sont reliés par un réseau multiaccès partagé (Tel que Fast Ethernet) et communiquent par diffusion de messages. Bien que la bande passante du réseau soit importante (de l'ordre de 100 Mbps), les systèmes ainsi construits ont une invariance à l'échelle limitée (25 à 100 noeuds).

Les systèmes basés switchs échangent des messages par routage à travers un réseau d'interconnexion pouvant avoir plusieurs topologies, allant des grilles simples aux hypercubes (voir figure 2.3). Les architectures en grilles, souvent présentées sur un seul circuit imprimé, conviennent pour les problèmes à deux dimensions (traitement d'images, théorie des graphes, etc.).

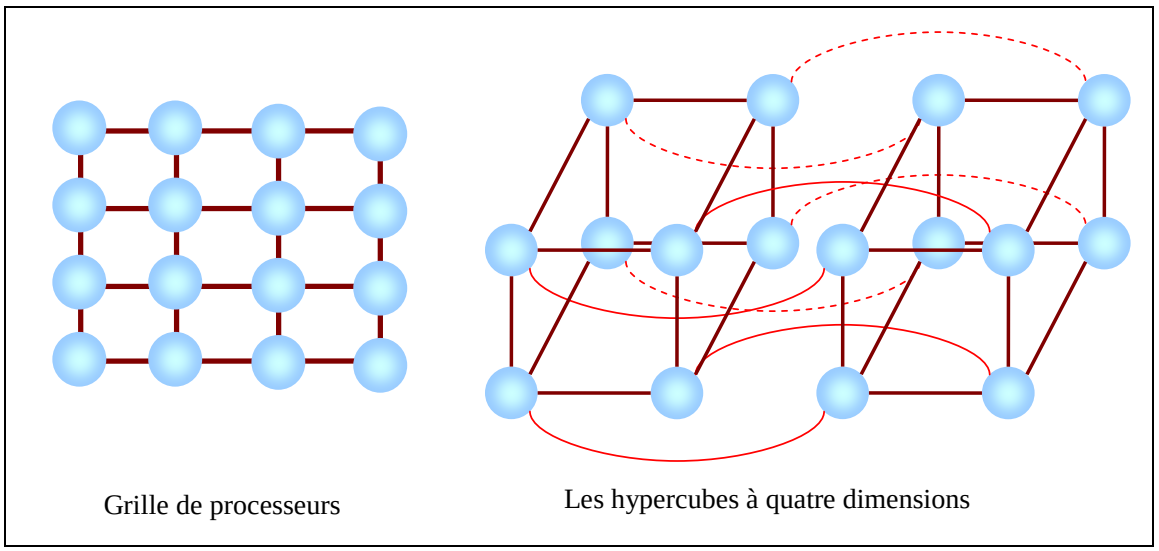

Figure 2.3. Exemples de systèmes basés switchs

Un hypercube est un cube à n-dimensions où chaque nœud (un processeur) est relié à n autres nœuds processeurs. Les hypercubes conviennent à la résolution de problèmes spécifiques tel que le calcul matriciel. Les systèmes basés switchs peuvent avoir des configurations très variées allant jusqu'à des superordinateurs massivement parallèles (MPP : Massively Parallel Processors) contenant des milliers de processeurs et dont le coût se chiffre en millions de dollars.

Les systèmes multiordinateurs hétérogènes sont les plus utilisés actuellement. Les ordinateurs peuvent avoir des structures et des performances nettement différentes et sont reliés par des réseaux hétérogènes.

2.3 TAXONOMIE AU NIVEAU SYSTEMES D'EXPLOITATION

Il existe une relation étroite entre les applications distribuées et les systèmes d'exploitation. Dans un premier lieu, la mise en œuvre des applications distribuées dépend des systèmes d'exploitation qui gèrent les différentes plateformes matérielles (i.e. les services qu'ils offrent). Dans un second lieu, les systèmes d'exploitation, eux-mêmes, peuvent être distribués (cas du système d'exploitation Chorus de Sun)

On peut diviser les systèmes d'exploitation en deux catégories : Les systèmes fortement couplés et les systèmes faiblement couplés. Dans le premier cas, le système d'exploitation essaye de maintenir une vue globale unique des ressources qu'il gère. Ce type de système à tendance à rendre la répartition physique transparente au niveau des applications. Dans le deuxième cas, on a affaire à une collection de plateformes ou chacune dispose de son propre système d'exploitation mais ces derniers coopèrent pour rendre leurs services et leurs ressources disponibles les uns aux autres, il s'agit des systèmes d'exploitation réseau.

Notons qu'il est important de signaler que les systèmes d'exploitation constituent, eux aussi, des applications distribuées. Cependant, contrairement à ces dernières, ils sont implantés directement sur les plateformes matérielles et en dépendent fortement.

2.3.1 Les systèmes d'exploitation distribués

On en distingue deux types :

- Les systèmes d'exploitation des plateformes multiprocesseurs (MPOS) (considérés comme des systèmes répartis particuliers)
- Les systèmes d'exploitation multiordinateurs (MCOS).

Les MPOS sont conçu pour supporter les hautes performances en utilisant des processeurs multiples. Un des buts des MPOS est de rendre le nombre des processeurs transparent pour les applications. Dans ces systèmes, les processus communiquent via l'utilisation de données situées dans des emplacements de mémoire partagés. La protection de ces emplacements est faite par des sémaphores ou des moniteurs.

Les MCOS ont une structure totalement différente et complexe par rapport aux MPOS. Ceci est du au fait que les structures de données communes ne peuvent être simplement placées dans une mémoire physique partagée. L'unique moyen de communication et l'envoi de messages (voir figure 2.4).

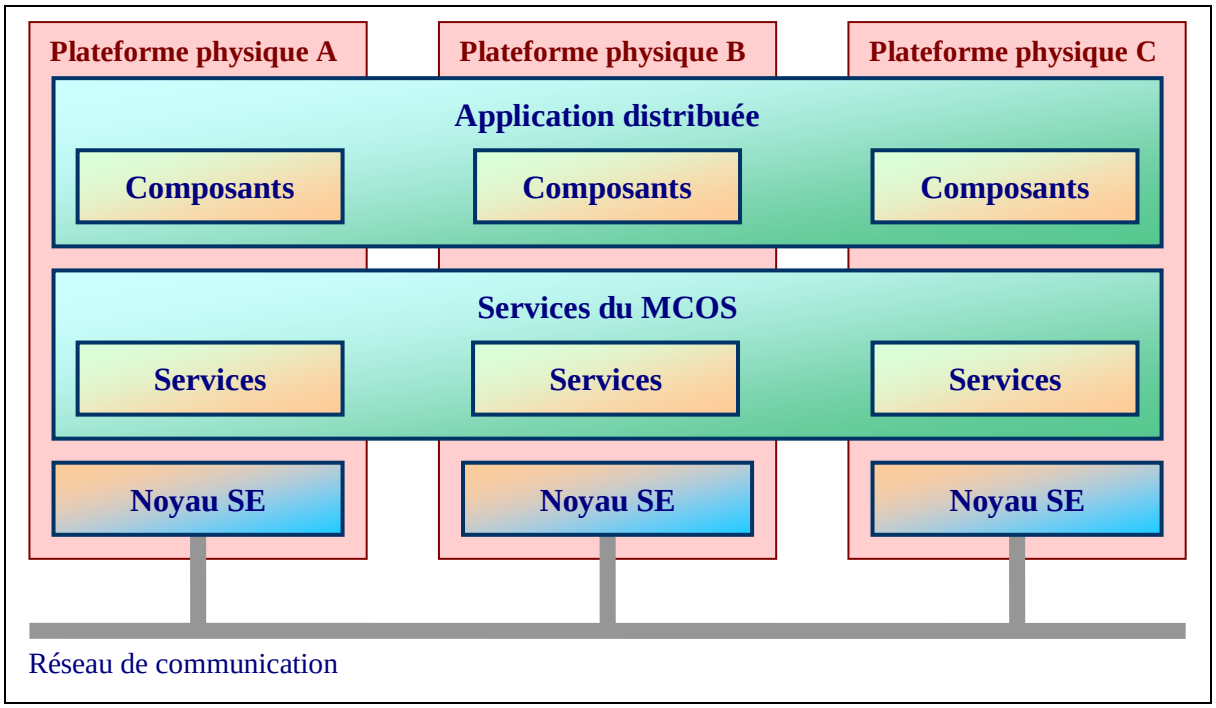

Figure 2.4. Positions des MCOS

La figure 2.4 peut être interprétée comme suit. Chaque nœud du réseau possède un système d'exploitation qui permet de gérer les ressources locales : mémoire, processeur, disque, … . De même, chaque nœud possède un module chargé de la communication entre plateformes (envoi et réception de messages). Sur chaque noyau

on greffe une couche commune qui implémente une machine virtuelle capable d'exécuter des tâches parallèles et concurrentes. Cette couche peut faire apparaître tout le système d'ordinateurs comme une machine multiprocesseur en implémentant une mémoire partagée. D'autres services sont assignés à cette couche : affecter une tâche à un processeur, masquer les pannes matérielles, assurer la transparence à la localisation et la communication interprocessus.

La programmation des systèmes MCOS est beaucoup plus difficile que la programmation des systèmes MPOS. La raison est que l'utilisation d'une mémoire partagée avec une protection par sémaphores ou moniteurs est plus simple que la manipulation des messages. Cette constatation est derrière la solution qui consiste à créer des MCOS en modifiant les MPOS par l'émulation d'une mémoire partagée virtuelle à partir de la mémoire virtuelle de chaque nœud.

Par exemple, on peut utiliser la pagination et avoir une mémoire répartie partagée basée sur la pagination. Les pages sont alors réparties sur tous les nœuds et on maintient une table globale des pages qui indique l'emplacement des pages sur les nœuds. C'est essentiellement la pagination classique excepté qu'au lieu d'utiliser le disque local, on utilise la mémoire virtuelle distante. Lorsqu'un processeur génère une référence d'une page qui n'est pas présente localement, un déroutement à lieu, le MCOS cherche la page et la ramène dans la mémoire locale au processeur ayant généré la référence, ce dernier pourra alors continuer son exécution. La figure 2.5 donne un exemple d'état d'une mémoire partagée virtuelle de 16 pages.

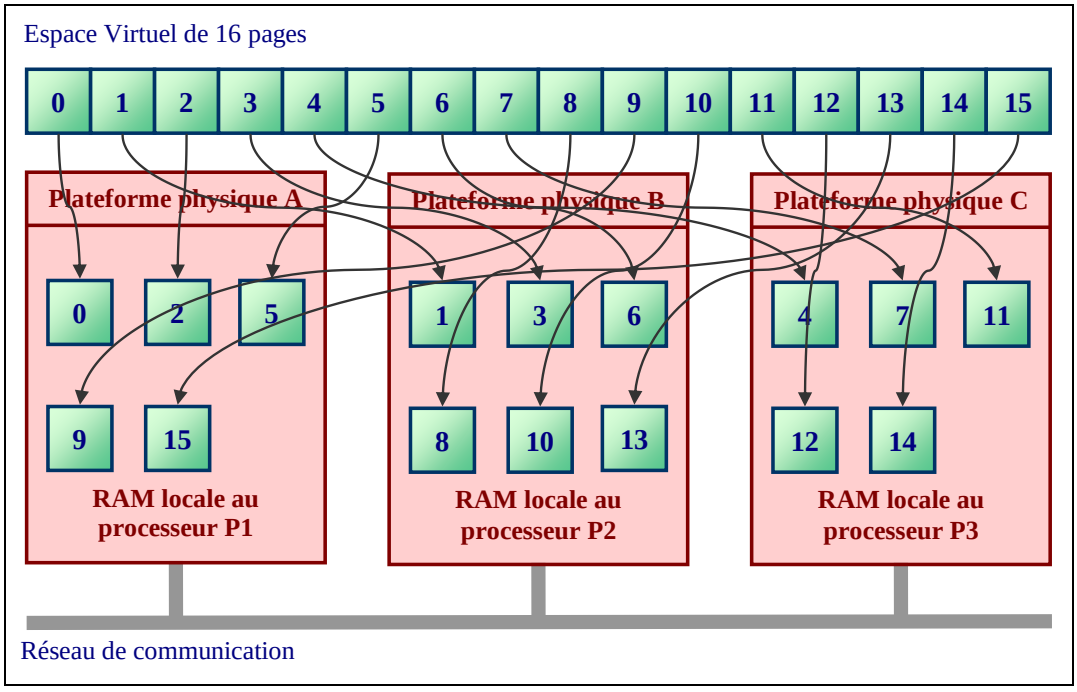

Figure 2.5. Une mémoire virtuelle partagée

Des améliorations peuvent être apportées à cette solution si on considère que certaines pages sont accédées uniquement en lecture et peuvent de ce fait être dupliquées.

2.3.2 Les systèmes d'exploitation réseau

Dans ces systèmes, on ne suppose pas que les plateformes matérielles sont homogènes et on ne cherche pas à faire apparaître l'ensemble comme un seul système. Il s'agit de réseaux où chaque nœud est une plateforme différente sur laquelle s'exécute un système d'exploitation différent (voir figure 2.6).

Les systèmes d'exploitation réseau offre divers services :

- Connexion avec une machine distante. Il s'agit de faire apparaître une machine comme un simple terminal d'une autre (c à d uniquement envoyer et afficher des caractères).
- Copie de fichiers d'un nœud à un autre (sans transparence).

Une amélioration courante de ces systèmes consiste à créer des serveurs qui cachent la répartition des fichiers sur les différentes machines et offrent un service de recherche et de transfert de fichiers aux machines qui sont alors considérées comme des clients.

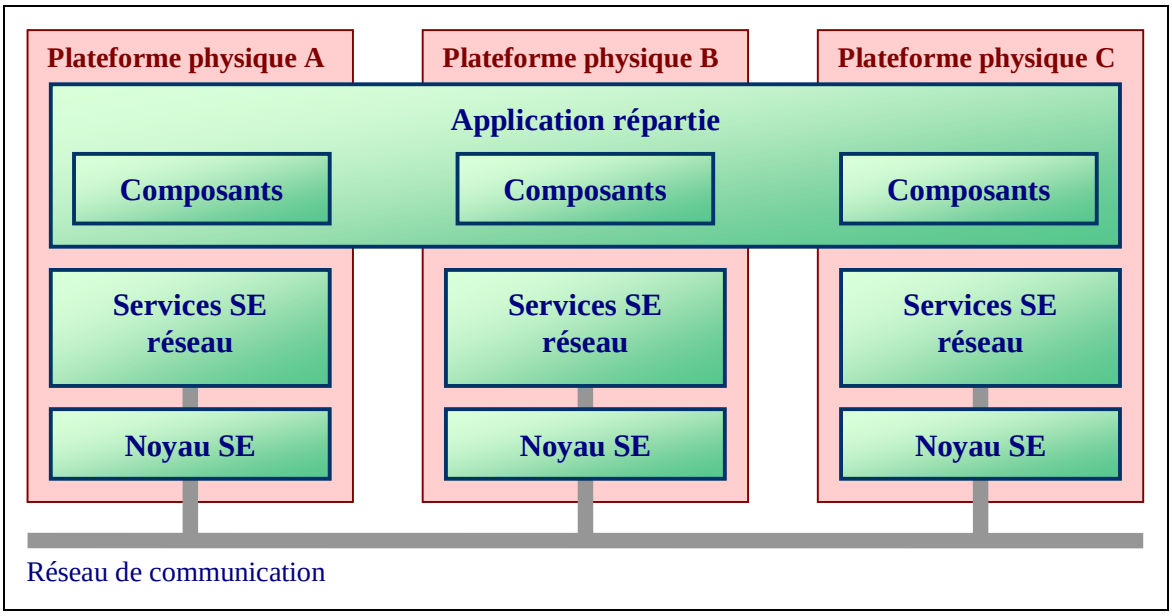

Figure 2.6. Les systèmes d'exploitation réseau

Un système d'exploitation réseau présente plusieurs inconvénients :

- Manque de transparence (connexion explicite, copie de fichiers d'une machine à l'autre explicitement désignées).
- Les machines étant indépendantes, elles sont gérées séparément (l'utilisateur d'une machine A à partir d'une machine B doit avoir un compte sur A et inversement, il faut alors gérer les mots de passes, les comptes et les droits accordés).

En contre partie, l'ajout d'une machine peut se faire librement et tout moment.

2.4 ARCHITECTURE DES APPLICATIONS DISTRIBUEES

Naturellement, les applications réparties ont plus d'indépendance vis-à-vis des plateformes physiques et peuvent de ce fait être organisées d'une multitude de façons. L'architecture client/serveur et ses variantes constituent actuellement les modèles le plus utilisés dans l'organisation des applications distribuées. Cependant d'autres modèles existent et leur utilisation augmente de jour en jour ; c'est le cas du modèle poste à poste (processus pairs) et ses variantes. Il n'est pas rare dans les applications distribuées que plusieurs modèles soient combinés à la fois pour tirer profit des avantages des uns et atténuer les inconvénients des autres. Dans la suite, nous présentons les différents modèles.

2.4.1 Architecture Client/serveur

C'est le modèle le plus utilisé et le plus important. Les processus représentant le système réparti, jouent les rôles de `client` pour un service donné et de `serveur` pour un autre. Par exemple, un navigateur Internet se comporte comme client lorsqu'il s'agit de récupérer une page Web. Un moteur de recherche est un serveur mais devient un client s'il déclenche d'autres moteurs de recherches sur d'autres sites Web. Actuellement, les moteurs de recherche typiques comportent plusieurs threads, certains servent les clients et d'autres se comportent comme client vis-à-vis des autres moteurs de recherche.

Dans le modèle Client/Serveur, on distingue deux sous modèles selon que le service est effectué par un ou plusieurs serveurs (voir figure 2.7). Dans ce dernier cas, plusieurs serveurs coopèrent pour exécuter une requête d'un client donné (e.g. Réservation de places d'avions pour une tournée impliquant plusieurs systèmes de réservation).

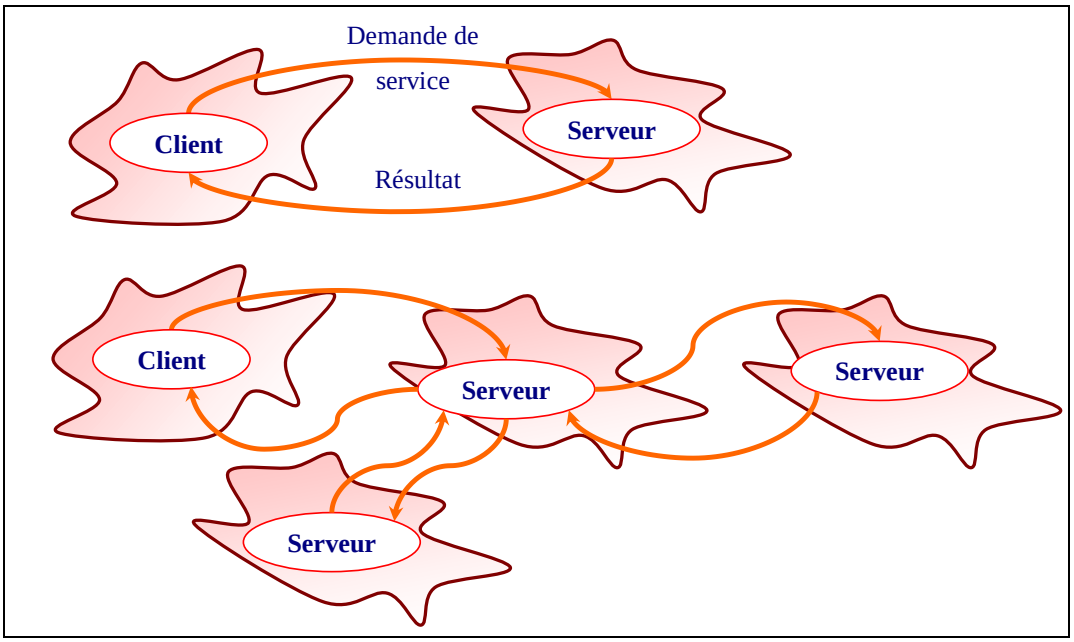

Figure 2.7. Un client/Un serveur Vs Un client/Plusieurs serveurs

2.4.1.1 Variantes de l'architecture Client/Serveur

Il existe diverses variantes du modèle Client/Serveur, nous les décrivons ci-après.

A. **Le modèle Client/Serveur et le code mobile (le serveur vient chez le client)** : Lorsqu'il s'agit de code mobile, le modèle Client/Serveur présente une particularité. En effet, au lieu que le serveur exécute un code et renvoi un résultat, il renvoi au client le code exécutable lui même (e.g. cas des applets Java). Ce modèle présente un avantage important, celui d'un temps de réponse meilleur, et un inconvénient important celui des risques que le code mobile engendre pour la sécurité du client. En effet, le code mobile peut être dangereux pour les ressources locales du client. Dans les cas des applets Java, les navigateurs ne les autorisent pas à accéder aux ressources locales du client.

B. **Le modèle Client/Serveur et les agents mobiles (le client va chez le serveur)** : Un agent mobile est un programme composé de code et de données, qui passent d'une plateforme à l'autre dans un réseau, dont le but de réaliser une tâche donnée. A l'arrivée, sur une plateforme donnée, l'agent invoque les services disponibles et utilise les ressources locales du serveur. Cette approche réduit le volume d'informations échangées sur un réseau et le temps de réponse. Les agents mobiles peuvent être utilisés pour :

   - La réalisation de tâches communes telles que comparaison d'offre de prix en visitant les sites des opérateurs économiques.
   - L'installation et la maintenance de logiciels
   - La répartition de charge où l'agent mobile migre vers les plateformes sous utilisées pour s'exécuter plus rapidement (solution testée au centre de recherche PARC de Xerox)

Comme pour le code mobile, les agents mobiles peuvent présenter un danger pour celui qui les accueille. Pour cela, il faut prévoir un accès restreint aux ressources et l'authentification de celui qui mandate l'agent (l'agent doit préserver secrètement l'information d'authentification). D'un autre coté, les agents mobiles sont vulnérables et ne peuvent continuer leurs tâches (survivre), si l'accès aux ressources locales leur est interdit.

C. **Le modèle Client/Serveur avec clients limités** : En général, le client dispose de données locales, d'un système d'exploitation et d'autres logiciels qui sont parfois difficiles à gérer et nécessite des utilisateurs qualifiés. Pour remédier à cela, on peut alléger le client de deux façons :

   - Client sans logiciels. La machine cliente ne dispose au départ que d'une application limité qui doit télécharger un système d'exploitation et les logiciels ou composants nécessaires à partir d'un serveur de fichier distant.

Les parties téléchargées s'exécutent sur la machine cliente mais les fichiers sont gérés par le serveur de fichiers distant.

- Client interface : Dans cette approche, le client dispose d'une couche logicielle qui se contente de jouer le rôle d'une interface d'affichage. Les applications sont exécutées sur le serveur qui doit être une machine multiprocesseur puissante.

Les approches qui visent à limiter les capacités des clients ont l'inconvénient de produire un système dont le temps de réponse est long spécialement lorsqu'il s'agit des applications de conception assistée par ordinateur. Notons que dans la pratique, il existe une panoplie de modèles ou le coté client d'une application peut avoir plus ou moins de responsabilité. Ceci est discuté dans les paragraphes suivants.

2.4.1.2 Architecture Client/serveur et répartition des rôles

Dans leur grande majorité, les applications visent le support de l'accès des utilisateurs à une base de données. De ce fait, on distingue trois niveaux différents dans une application Client/Serveur :

- Niveau Interface utilisateur. La partie cliente dans une application implémente, souvent, l'interface utilisateur. Ce niveau permet de gérer l'interaction de l'utilisateur avec l'application. L'interface utilisateur peut être très simple comme elle peut être très sophistiquée. Dans le cas ou l'interface ne fait que gérer l'affichage de caractères et leur saisie par un clavier, l'appellation Client/Serveur n'est pas très appropriée. Actuellement, la majorité des applications assignent au clients au moins l'affichage graphique et diverses fonctionnalités utilisant la souris.
- Niveau Données. Le niveau Données dans une application Client/Serveur contient les programmes qui maintiennent les données de l'application. Les données à ce niveau sont persistantes. Dans le cas le plus simple il s'agit d'un système de fichiers, mais, souvent, c'est des bases de données complètes qui matérialisent le niveau Données. Dans le cas le plus général du modèle client/serveur, les données sont du coté du serveur.
- Niveau Traitement de l'application dit aussi logique métier. Le niveau traitement dépend des applications. Il consiste en un ensemble de fonctionnalités qui se situent entre le niveau Interface et le niveau Données.

La figure 2.8 illustre ces niveaux à travers un exemple. Il s'agit de l'accès à un moteur de recherche à travers le WEB. Le client n'est autre qu'un navigateur Internet qui affiche les pages Web et transmet les requêtes de l'utilisateur en utilisant le protocole HTTP.

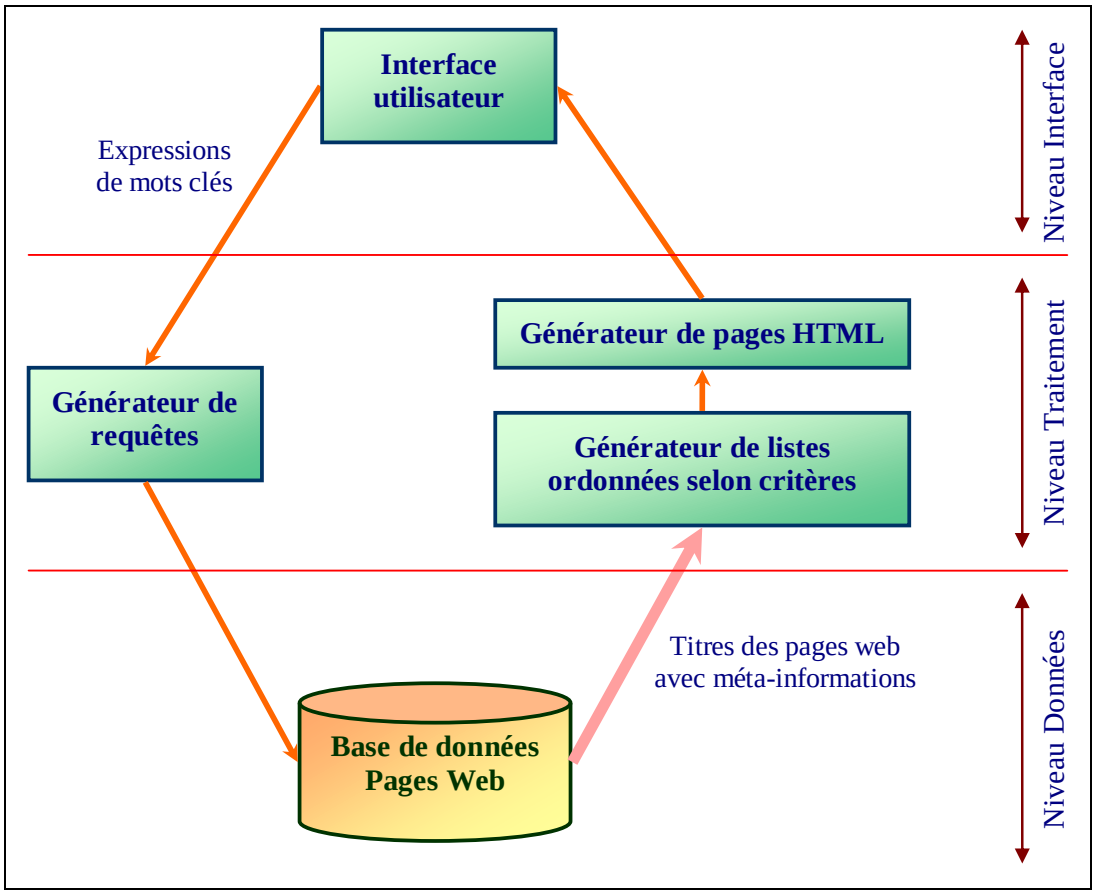

Figure 2.8. Un moteur de recherche sur le Web

2.4.1.3 Architectures multitiers

La répartition des rôles discutés précédemment, suggère différentes possibilités de répartition d'une application qualifiées de répartition multitiers (Multitiered en anglais).

Le terme Client/Serveur a été traditionnellement associé à la configuration matérielle qui consistait en un microordinateur connecté à un serveur SQL de base de données. Le terme désigne, donc, un modèle de partage ou les tâches sont réparties entre des couches clientes et serveurs dites tiers.

L'architecture *un tier* correspond aux applications classiques des ordinateurs centraux (mainframe) où des terminaux permettent à des utilisateurs d'interagir avec une application monolitique qui effectue des traitement (logique métier), gère des bases de données et communique avec l'utilisateur.

Dans les architectures *deux tiers*, l'application est partagée en deux parties qui, naturellement, résident sur deux plateformes distinctes. Le client accède directement au serveur de la base de données et les traitement peuvent être du coté client comme du coté serveur de la base de données sous forme de procédures.

Lorsque les traitements ont lieu du coté client, on a affaire à des clients lourds (fat client). Le serveur gère la base de données et reçoit des requêtes SQL qu'il exécute et renvoi un ensemble de données résultats au client.

Plus l'application est importante, plus le client est lourd ce qui nécessite une plateforme support performante ce qui est un inconvénient important.

Lorsque les traitements résident du côté du serveur, on parle de serveur lourd. Les clients ne font qu'invoquer les procédures stockées dans le serveur de bases de données. Du point de vue performances, la configuration où le serveur est lourd est meilleure que la précédente car la communication entre client et serveur est moins importante.

Entre client lourd/léger et serveur lourd/léger, il existe une multitude de variantes que nous illustrons par le schéma de la figure 2.9.

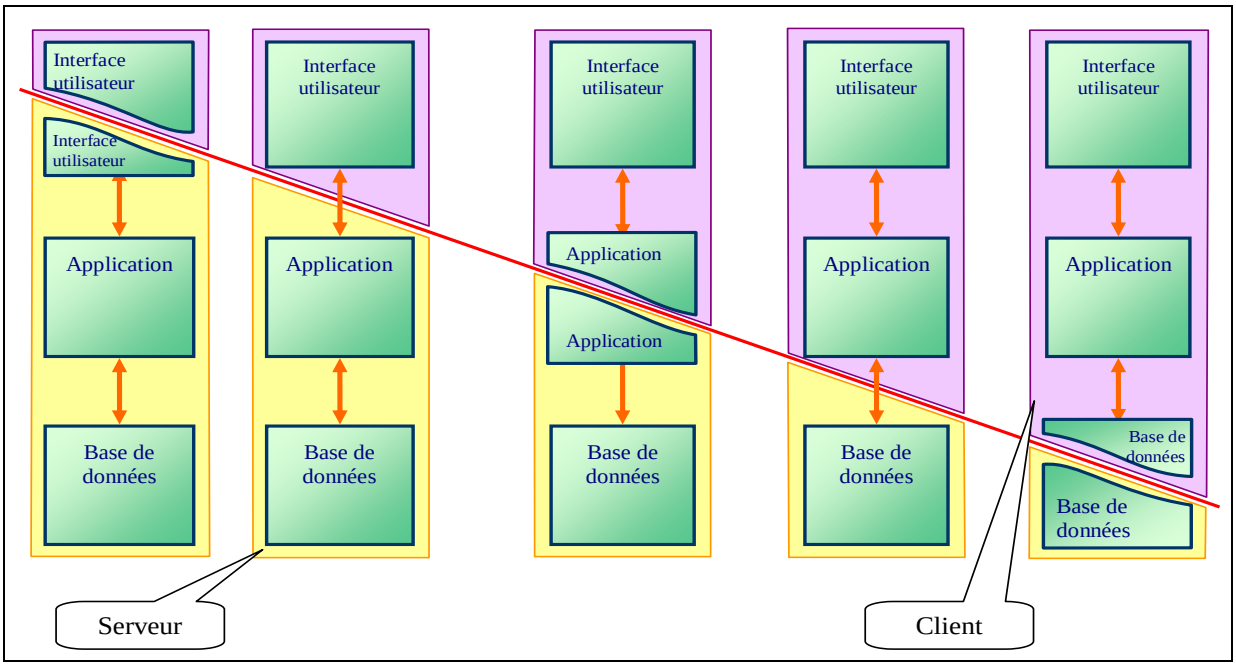

Figure 2.9. Architecture deux tiers

Dans cette figure, on constate l'existence de cinq cas :

1- La plateforme du coté client joue le rôle de terminal

2- L'application contient un minimum concernant l'interface. L'interface est une couche graphique qui communique avec l'application

3- Une partie de l'application est dans la plateforme cliente. Par exemple vérification des formulaires (consistance) ou édition de textes sur la plateforme cliente et des fonctions avancées sur le serveur.

4- Configuration très courante. Le client est un microordinateur ou une station connecté à travers un réseau à une base de données ou un système de fichiers. Toute l'application s'exécute sur la plateforme cliente mais les opérations sur la base se font sur le serveur.

5- Ce cas correspond à un maintien local au client d'une partie des données.

Généralement l'architecture deux tiers n'est pas invariante à l'échelle. Au-delà d'une centaine de clients, les performances chutent considérablement.

Dans les configurations plus récentes, un *tiers du milieu* est ajouté entre le client et le serveur de la base de données. On a alors affaire à des architectures *trois tiers* où le premier tiers n'est autre qu'un client léger qui implémente une logique de présentation, le second tiers dit *serveur d'application*, implémente la logique métier et le troisième est un serveur de bases de données. Dans un environnement donné, il est possible de trouver plusieurs serveurs d'application et plusieurs serveurs de bases de données.

Le tier du milieu implémente, en plus de la logique métier, des opérations de translation qui convertissent les demandes des clients en requêtes de base de données et convertissent les résultats de ces requêtes selon le format utilisé par le client.

Souvent, le tiers client interagit avec le serveur de l'application moyennant un protocole standard tel que le RPC (Remote Procedure Call) ou HTTP. A son tour, le tiers du milieu interagit avec le serveur de bases de données en utilisant un protocole standard tel que SQL, ODBC JDBC.

Cette configuration, permet une meilleure invariance à l'échelle et l'utilisation de protocoles standards permet de créer une indépendance entre les tiers ce qui permet à chacun d'évoluer plus facilement : évolution de la logique métier, changement de la base de données en choisissant un autre fournisseur sans altétérer les autres tiers, etc.

L'architecture multitiers utilise plusieurs tiers du milieu au lieu d'un seul tiers milieu lourd. Ces architectures sont actuellement très utilisées est permettent de supporter des applications volumineuses en les décomposant en couches plus simples à développer et à maintenir. N'importe quelle application client/serveur peut être implémentée avec une architecture multitiers où la logique métier est décomposée en plusieurs serveurs.

Comme avantages de l'architecture multitiers, on cite :

- Chaque tiers est plus ou moins indépendant des autres ce qui lui permet d'évoluer sans trop de contraintes. En effet, la concentration de la logique métier dans les tiers du milieu permet la modification de celle-ci sans

nécessiter le changement d'une multitude de tiers clients qui peuvent être physiquement éloignés.

- Réduction importante du volume d'informations échangé sur le réseau (on échange uniquement les informations nécessaires à la réalisation d'un service), ce qui améliore les performances.
- La base de données peut être partagée par plusieurs utilisateurs ayant chacun sa logique métier. Ceci est possible par l'utilisation de plusieurs serveurs d'applications différents.
- Possibilité de répartir la charge du tiers milieu sur plusieurs plateformes physiques.
- Possibilité de dupliquer les serveurs de l'application et les serveurs des bases de données.
- Développement et maintenance plus aisée des applications distribuées.

L'utilisation d'une architecture trois/multitiers n'exclu pas les architectures un/deux tiers. Si pour une application réduite, un ou deux tiers sont convenables, pour une application importante l'architecture multitiers conviendrai davantage.

2.4.2 Le modèle du Mandataire/Cache

Un cache est un espace mémoire qui maintient une copie des objets récemment utilisés proches, vis-à-vis du client, que les objets originaux. Un objet reçu est ajouté au cache remplaçant éventuellement un objet existant. Lorsqu'un client demande un objet, le gestionnaire du cache essaye d'abord de le trouver dans le cache et le transmettre au client. Si l'objet n'est pas dans le cache, le gestionnaire transmet la demande au serveur qui détient l'objet.

Le cache peut être géré par le client lui-même comme il peut être géré par un gestionnaire indépendant dit Serveur Mandataire (appelé aussi Proxy). Dans ce dernier cas, le cache peut être utilisé par plusieurs clients (voir figure 2.12).

Par exemple, dans le Web, les mandataires maintiennent des caches des pages récemment visitées mais avant de livrer une page à un client, le mandataire vérifie au moyen d'une requête spéciale, du protocole HTTP, si la page qu'il a est conforme à l'originale.

Les mandataires permettent d'augmenter les performances en diminuant le temps de réponse. Les mandataires peuvent implémenter un protocole de sécurité tel que les pare-feu (Firewall). Le modèle du mandataire est souvent utilisé comme modèle complémentaire avec d'autres modèles (i.e. combiné avec les autres).

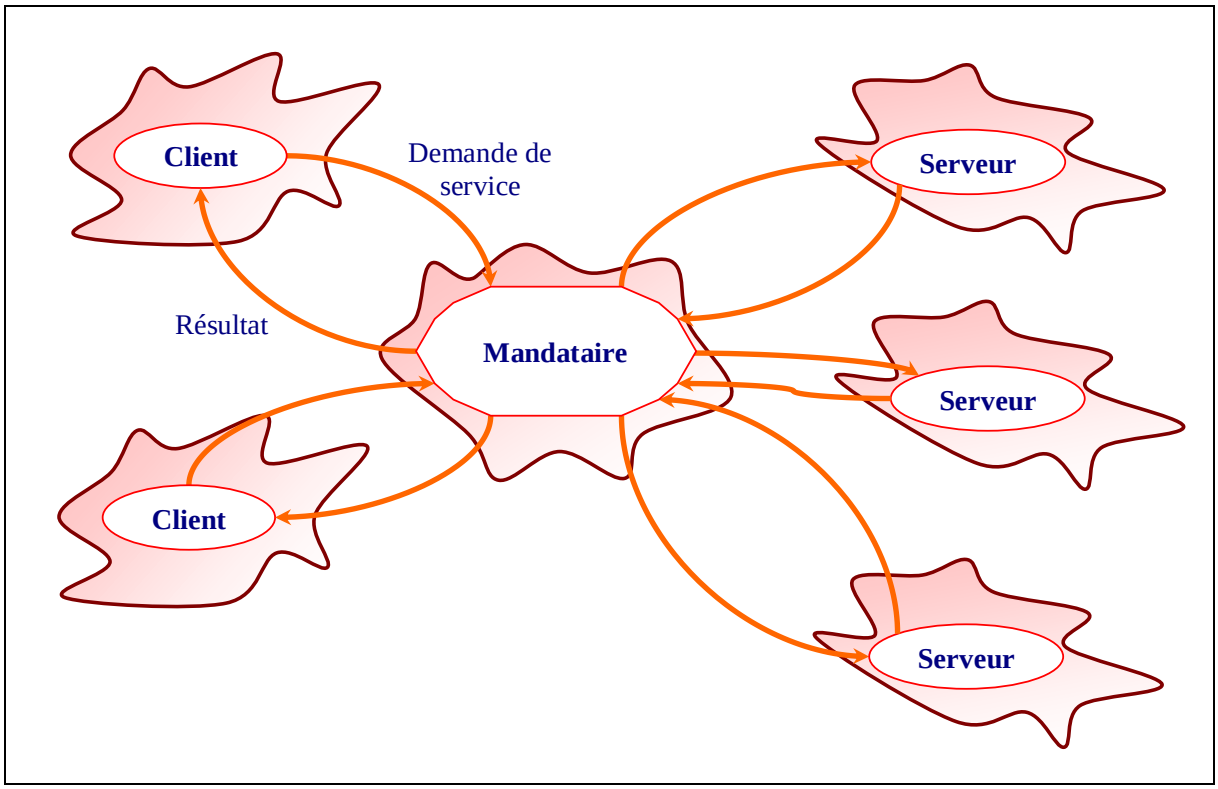

Figure 2.12. Le mandataire comme intermédiaire entre clients et serveurs

2.4.3 Architecture pair-à-pair (Peer To Peer)

Dans ce type d'architecture, il n'existe pas de distinction, en terme de clients et de serveurs, entre les composants (processus) d'un système distribué. Les processus jouent des rôles similaires et coopèrent d'égal à égal pour réaliser une activité répartie (Voir figure 2.13).

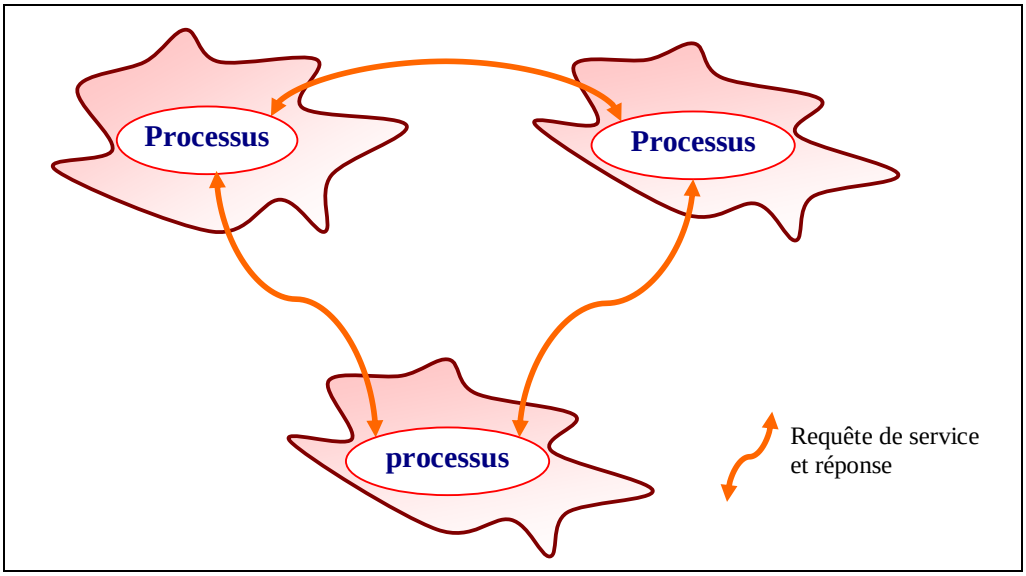

Figure 2.13. La coopération des processus pairs

Le terme *Peer-to-peer (abrégé en P2P)*, qu'on peut traduire par *pair-à-pair (poste à poste ou encore égal à égal)* désigne tout système où les participants (les pairs) mettent en partage des ressources locales (qui peuvent être des capacités de traitement, des fichiers, des espaces de stockage, des moyens de communication, etc.) sans utilisation de serveurs spécifiques.

Les participants partages les ressources locales en établissant des communications directes entre eux moyennant les protocoles TCP/IP. Ainsi, chaque participant est à la fois un client et un serveur. Il est *serveur* de ce qu'il possède et souhaite partager et *client* de ce que les autres mettent à sa disposition. Le P2P désigne donc une classe d'applications qui tirent profit des ressources matérielles ou humaines qui sont disponibles sur le réseau Internet.

L'infrastructure support des applications peer-to-peer comprend principalement des réseaux utilisant les protocoles TCP/IP, protocoles symétriques qui ne font aucune distinction entre client et serveur, et des composants logiciels particuliers qui remplissent, à la fois, les fonctions de client et de serveur. Ces derniers sont parfois appelé *servents* (de la contraction de serveur et de client, due à Gnutella), ou, plus communément mais de façon réductrice, *clients*. Les servents peuvent matérialiser toute l'application, et sont alors en interaction directe avec l'utilisateur, comme ils peuvent ne constituer qu'une couche offrant des services à diverses applications. La figure 2.14 donne un aperçu sur l'infrastructure décrite.

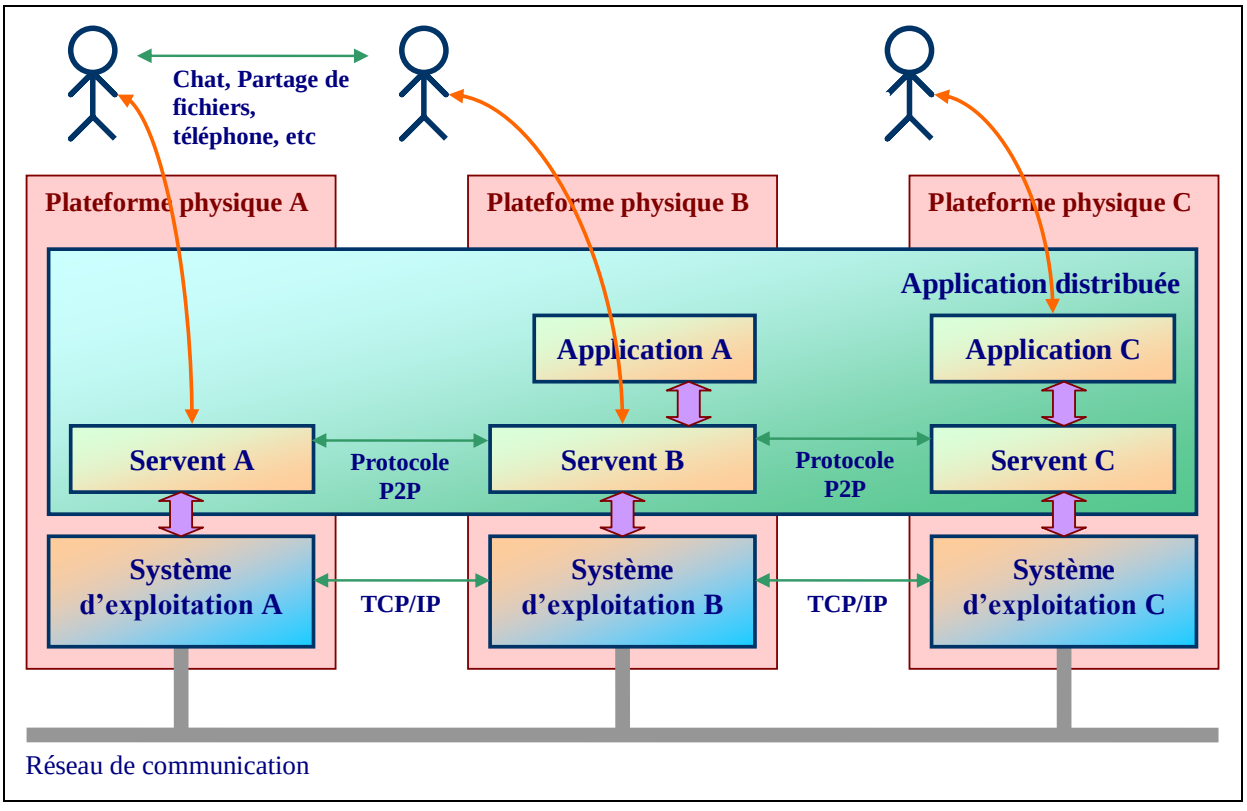

Figure 2.14. Infrastructure des systèmes P2P

2.4.3.1 Avantages des systèmes Pair-à-pair

Bien que l'utilisation dominante, des systèmes P2P, soit, actuellement, le partage de fichiers, le modèle pair-à-pair va bien plus loin que ce simple partage. En effet, il est possible de décentraliser des services et de mettre à la disposition des autres participants des ressources. Vu que chaque utilisateur d'un réseau pair-à-pair peut proposer des ressources et en obtenir, les systèmes pair-à-pair permettent de faciliter le partage d'informations et rendent la censure ou les attaques des services plus difficiles.

En général, les systèmes P2P présentent certains avantages par rapport aux systèmes client-serveur :

- Réduction des coûts. Au lieu d'acquérir un nouveau matériel pour construire une infrastructure client-serveur, il est possible d'utiliser les plateformes existantes avec une approche P2P ce qui évite des dépenses supplémentaires et réduit le coût d'administration. A titre d'exemple, un réseau de stockage P2P évite le recours aux serveurs de stockages centraux tout en offrant des performances meilleures.
- Invariance à l'échelle (passage à l'échelle): La distribution/duplication des ressources et services permet de répartir l'information sur l'ensemble du réseau P2P et évite l'apparition de goulots d'étranglements. Ce qui permet une bonne invariance à l'échelle. Par exemple, certains systèmes d'échange de fichiers tel que Napster (échange de fichiers MP3) pouvaient servir plus de six millions d'utilisateurs simultanément.
- Réseaux ad hoc : L'approche P2P est convenable pour les réseaux ad hoc où la connexion est intermittente. L'utilisateur est libre de se connecter ou se déconnecter au réseau pour des durées quelconques. L'approche P2P est ainsi convenable pour des applications où un contrôle centralisé n'est pas envisageable. Cas de l'informatique diffuse, l'informatique mobile, etc.
- La fiabilité : Vu que le bon fonctionnement d'un système P2P ne dépend pas d'un participant spécifique mais donne une importance égale à chaque chacun des participants, la fiabilité se trouve améliorée. En effet, la panne d'un nœud n'a pas d'influence sur le système dans son ensemble, ce qui n'est pas le cas lors de la panne d'un serveur dans une architecture client-serveur.

Ces avantages font des systèmes pair-à-pair des outils prévilegiés pour décentraliser des services devant avoir une haute disponibilité et des coûts de maintenance faibles. Il s'agit des services telsque : la diffusion de données multimédia, la distribution des logiciels et leurs mises à jour, la communication téléphonique et messagerie, la gestion des noms des domaines (DNS), etc.

2.4.3.2 Les inconvénients des systèmes P2P

L'approche P2P présente des inconvénients importants aussi. En effet, les problèmes de sécurité ou de comptabilité sont plus simples à résoudre dans un système doté d'un serveur central. De même, la disponibilité des ressources n'est pas toujours garantie lorsque les participants aux systèmes P2P sont peu nombreux. La rupture de la connexion par un participant peut rendre une ressource non accessible si elle n'est pas disponible chez d'autres participants. C'est typiquement le cas dans l'échange de fichiers.

2.4.3.3 Les variantes du modèle P2P pour le partage de fichiers

Les variantes des architectures P2P peuvent être classées en quatre catégories : architecture centralisée, architecture décentralisée, architecture hiérarchique et architecture en anneau. Dans la pratique, ces architectures sont souvent combinées pour avoir des systèmes distribués P2P hybrides.

A- Architecture centralisée : le peer-to-peer assisté

Dans une architecture centralisée, il doit y exister un serveur qui se charge de mettre en relation directe tous les participants connectés. Le serveur maintient une base de donnée centrale qui consiste en un index de tous noms des fichiers, que chaque participant met à la disposition des autres, couplés avec les adresses IP des participants qui les possèdent. La base ne contient à aucun moment les fichiers eux-mêmes mais seulement leurs intitulés. La mise à jour de la base se fait à chaque fois dès qu'un participant se connecte ou se retire du réseau. La figure 2.15 illustre cette architecture.

Pour télécharger un fichier, l'utilisateur soumet à l'aide du servent une requête comportant les mots clés pouvant apparaître dans la nom du fichier. Le serveur répond avec une liste de noms accompagnés des adresses IP correspondantes. L'utilisateur dispose alors pour chaque fichier d'une ou plusieurs adresses IP. Il ne lui reste qu'à établir (en utilisant son servent) une connexion directe avec le servent possédant le fichier recherché et le télécharger.

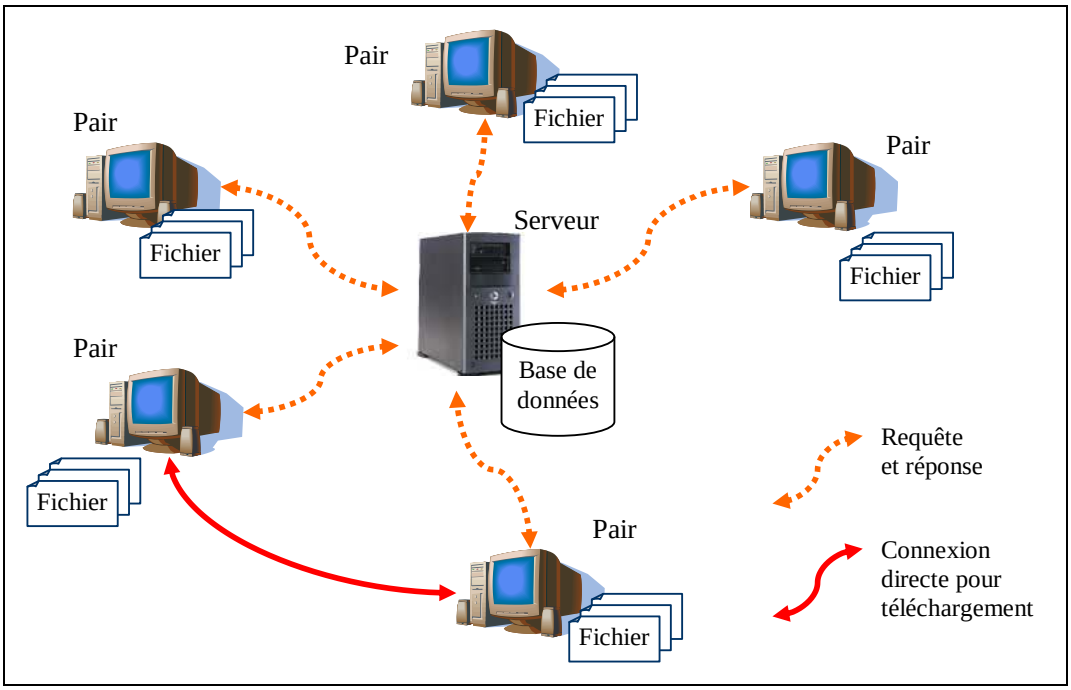

Figure 2.15. Architecture centralisée

Comme pour l'architecture client-serveur, le point faible de l'architecture P2P centralisée réside au niveau du serveur. En effet, en cas de panne de ce dernier aucun échange ne peut avoir lieu. De même, le serveur peut être souvent sollicité ce qui réduit les performances (bande passante du serveur). La présence des adresses IP sur le serveur ne permet pas un échange totalement anonyme des ressources.

B - Architecture en anneau

Dans le but d'accroître la fiabilité de l'architecture centralisée, le serveur central est remplacé par un anneau virtuel de serveurs. La figure 2.16 donne un aperçu sur cette architecture.

Chaque serveur de l'anneau maintien des informations sur les participants qui lui sont connectés et échange ces informations avec les autres serveurs. Ainsi, en cas de panne d'un serveur le système reste opérationnel. Avec une telle approche, on constate une meilleure disponibilité des serveurs et une répartition de la charge qui préserve les performances (répartitions des demandes de connexions et requêtes qui préserve la bande passante. L'architecture en anneau est souvent utilisée lorsque les machines sont relativement proches (appartenant souvent à la même organisation).

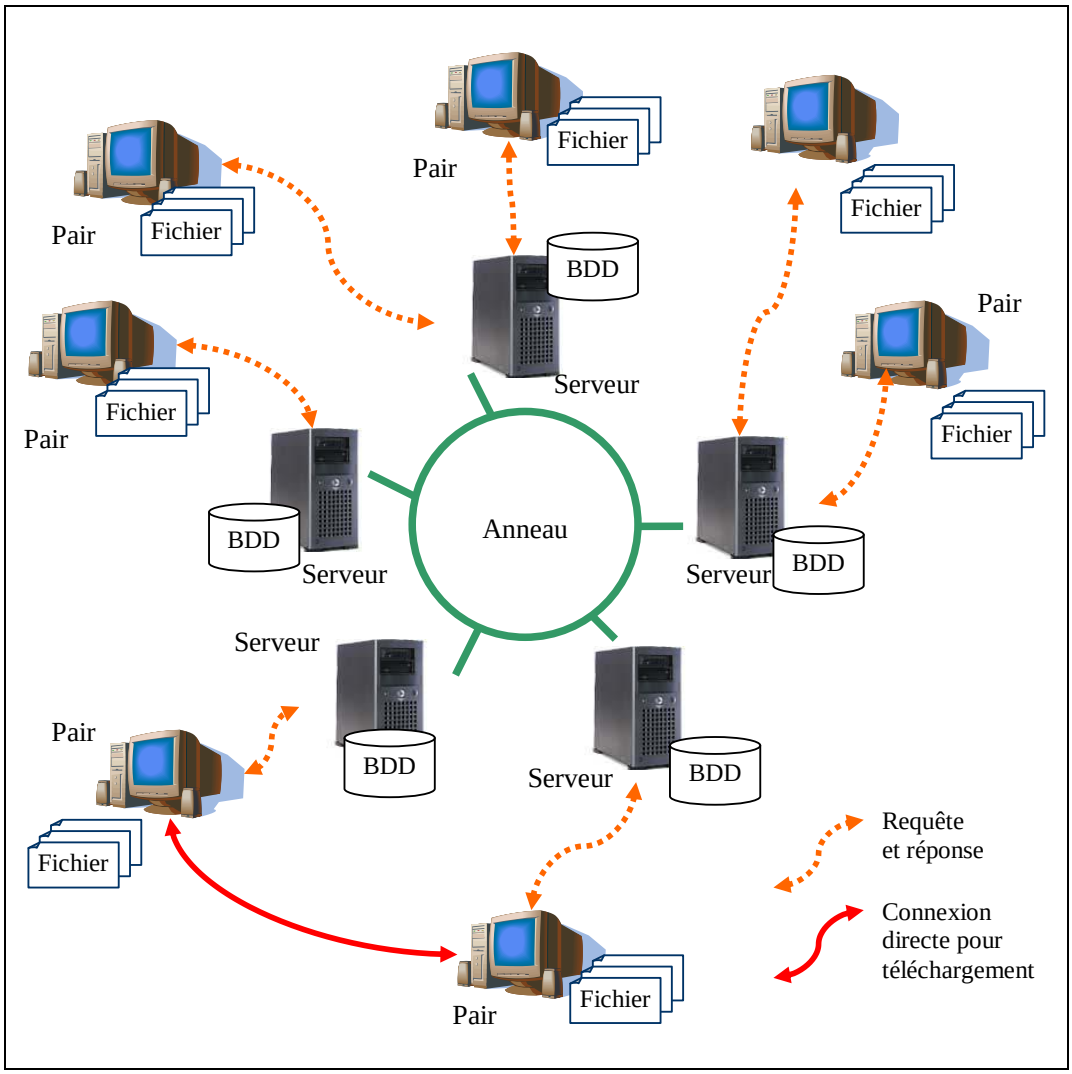

Figure 2.16. Architecture P2P en anneau

C – Architecture hiérarchique : Au lieu d'organiser les serveurs en anneau, l'architecture hiérarchique élabore plutôt une hiérarchie qui réduit le volume d'informations échangées entre les serveurs, ce qui préserve la bande passante. La figure 2.17 illustre ce principe.

D – Architecture décentralisée (Le P2P pur) : Dans ce type d'architecture, il n'existe pas de serveurs. Tous les participants ont le même statut. Pour rejoindre le réseau, un participant P doit d'abord diffuser un message spécial sur le réseau physique. Les pairs directement proches de P et qui reçoivent ce message renvoient leurs adresses IP. Vu qu'il n'y a pas de serveurs, les requêtes de P seront diffusées à ces voisins et ses derniers les diffusent à leurs voisins immédiats et ainsi de suite. Ce qui engendre un trafic important sur le réseau physique et réduit ainsi les performances. Les architectures décentralisées présentent l'avantage d'être insensibles aux pannes des pairs. La figure 2.17 illustre l'approche.

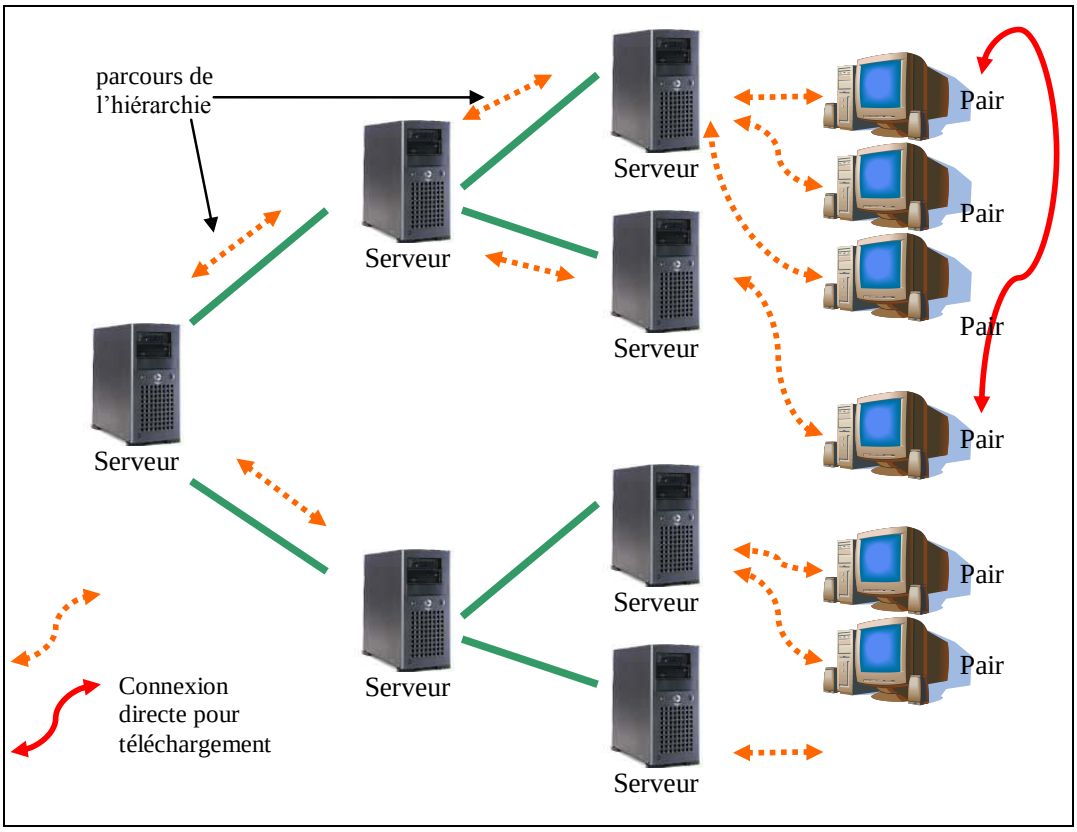

Figure 2.16. Architecture P2P hiérarchique

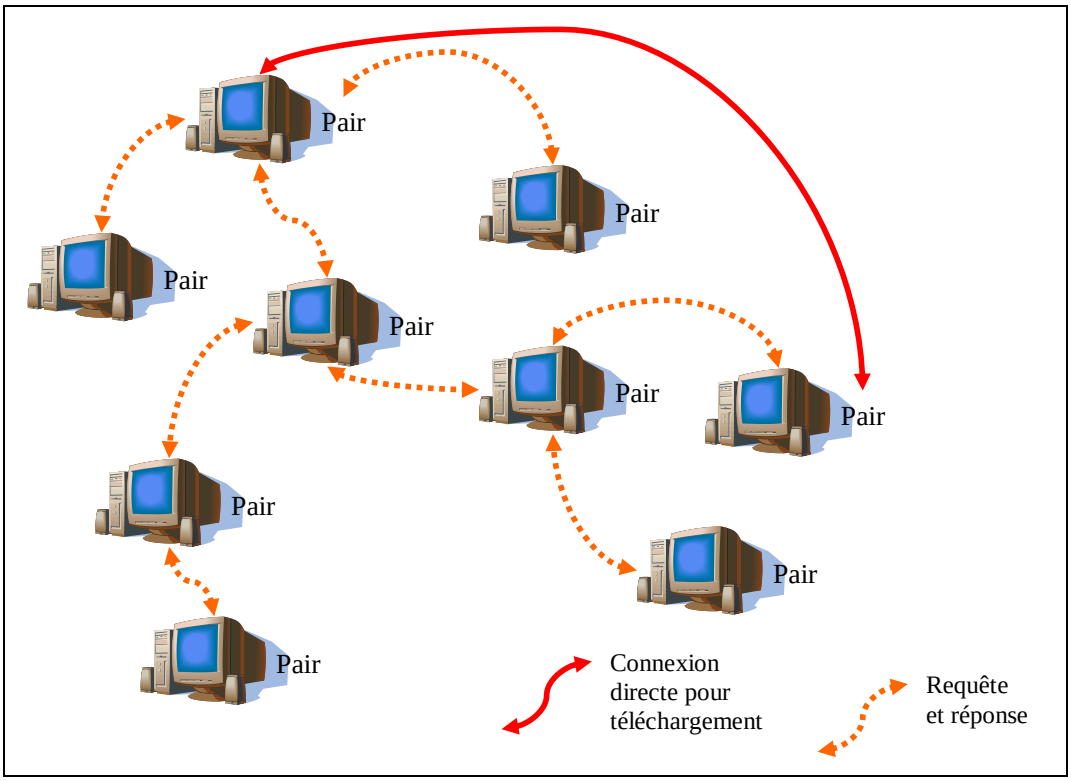

Figure 2.17. Architecture P2P décentralisée

2.4.3.4 Quelques exemples de systèmes

e-Mule : e-Mule est la version actuelle de e-Donkey qui est né en septembre 2000. Il adopte une architecture centralisée avec une multitude de serveurs et permet le transfert de tous types de fichiers en utilisant des servents pour les plateformes windows et Unix.

Chaque utilisateur peut créer son propre serveur et les serveurs sont reliés entre eux. Lors de sa connexion avec l'un des serveurs, le pair fournit la liste des fichiers qu'il souhaite mettre à la disposition des autres. Après connexion, le pair a la possibilité de soumettre une requête de recherche au serveur auquel il est connecté ou à tous les serveurs dont les adresses lui sont fournies par les autres pairs. Les fichiers partagés ne transitent pas par les serveurs, ces derniers se contentent de maintenir des index composés de noms de fichiers et d'adresses IP.

e-Mule utilise au niveau bas le protocole UDP et au niveau des servents le protocole MFTP (Multisource File Transfert Protocol) qui est une variante du FTP où les fichiers sont décomposés en blocs et les téléchargements se font par blocs. Il est ainsi possible de télécharger les blocs d'un fichier de différentes sources, ce qui améliore grandement les performances.

Gnutella : Il s'agit d'un réseau P2P décentralisé qui a évolué au fil du temps d'une architecture semblable à celle de e-Donkey vers une architecture P2P pure qui se caractérise par l'absence de serveurs. Pour se connecter au réseau, le servent diffuse un message particulier sur Internet (dit PING) pour récupérer les adresses IP et quelques informations sur les données partagées des servents les plus proches (Ces derniers renvoient des messages dits PONG). Une fois la connexion établie (Récupération des adresses IP), le servent envoi des trames de recherche aux différents pairs et récupère, par des trames réponses, les noms des fichiers partagés. Il ne reste plus qu'à télécharger directement le fichier sélectionné du pair qui le détient. Comme pour e-Mule, Gnutella utilise le protocole MFTP.

Divers servents Gnutella existent pour les plateformes Windows, Unix et Macintosh :

- **Pour Windows** : Gnutella Clients, BearShare, Gnucleus, Morpheus, Shareaza, Swapper, XoloX, LimeWire, Phex
- **Pour Unix et Linux** : Gnutella Clients, Gnewtellium, Gtk-Gnutella, Mutella, Qtella, LimeWire, Phex
- **Pour Macintosh** : Gnutella Clients, LimeWire, Phex