Chapitre 2
Architectures et modèles des systèmes distribués

1. Introduction

L'objectif de ce chapitre est de présenter brièvement les modèles et les architectures des systèmes distribués. En particulier, nous présentons les modèles de type un tiers, deux tiers et trois tiers, et une synthèse approfondie sur les architectures logicielles n-tiers. On abordera les technologies utilisées et les moyens à mettre en œuvre. Nous donnons des explications des termes clients légers, clients lourds et une étude sur les avantages et les inconvénients des différentes architectures.

2. Les niveaux fonctionnels d'une application répartie

En règle générale, une application distribuée peut être découpée en trois niveaux d'abstraction distincts (figure1).

- **Couche de présentation :** Cette couche représente la partie cliente de l'application. Elle permet à l'utilisateur d'interagir avec l'application. L'interface utilisateur peut être très simple comme elle peut être très sophistiquée. Actuellement, la majorité des applications assignent aux clients au moins l'affichage graphique et diverses fonctionnalités utilisant la souris
- **Services métier :** Décrivant le niveau traitement de l'application dit aussi logique métier. Ils peuvent être découpés en deux familles :
  - Locaux : regroupant les contrôles effectués au niveau du dialogue avec la couche présentation, visant essentiellement le contrôle et l'aide à la saisie.
  - Globaux : constituant l'application elle-même. Cette couche, appelée Business Logic ou couche métier, contient les règles internes qui régissent une entreprise donnée [6].

  Le niveau services métier consiste en général en un ensemble de fonctionnalités qui se situent entre le niveau Interface et le niveau Données.
- **Persistance :** Représente le niveau de données de l'application, regroupant l'ensemble des mécanismes permettant la gestion des informations stockées par 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.

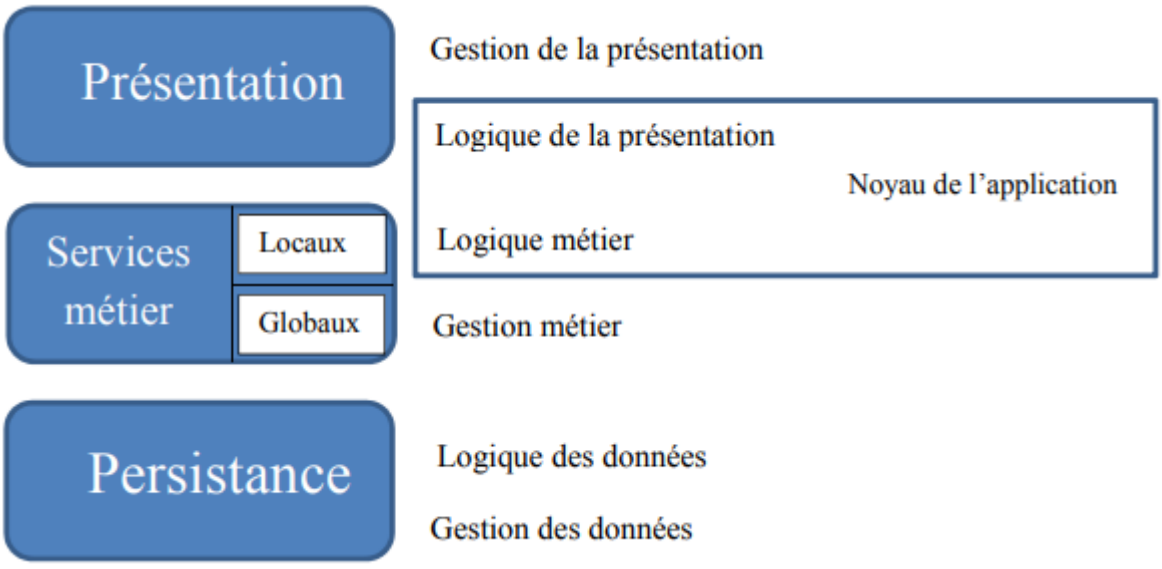

Figure1 : Couches fonctionnelles d'une application répartie

Ces trois niveaux peuvent être imbriqués ou répartis de différentes manières entre plusieurs machines physiques.
Le noyau de l'application est composé de la logique de l'affichage et la logique des traitements. Le découpage et la répartition de ce noyau permettent de distinguer les architectures applicatives du système distribué.

3. Architecture des systèmes distribués

D'une manière générale, une architecture distribuée désigne un système d'information pour lequel l'ensemble des ressources disponibles ne se trouvent pas au même endroit ou sur la même machine.

L'architecture client/serveur et ses variantes constituent les modèles le plus utilisés dans l'organisation un système distribué. Cependant d'autres modèles existent le modèle poste à poste (processus pairs) et ses variantes. En autre, 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 l'analyse des solutions architecturales au sein des systèmes modernes, il est essentiel que le développeur ait l'opportunité d'évaluer la fiabilité architecturale de son application, en tant que partie importante du système [4-8]. Il existe des fonctionnalités spécifiques dans la conception de systèmes informatiques distribués. Tout d'abord, c'est la dépendance du modèle architectural vis-à-vis de certaines exigences non fonctionnelles du système telles que les performances, la sécurité, la sûreté, la fiabilité [3].

3.1 Architecture Client/serveur

La particularité de nombreuses applications «client-serveur» des systèmes distribués modernes est leur diversité et leur dispersion [1], [2]. Le modèle Client/Serveur 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.

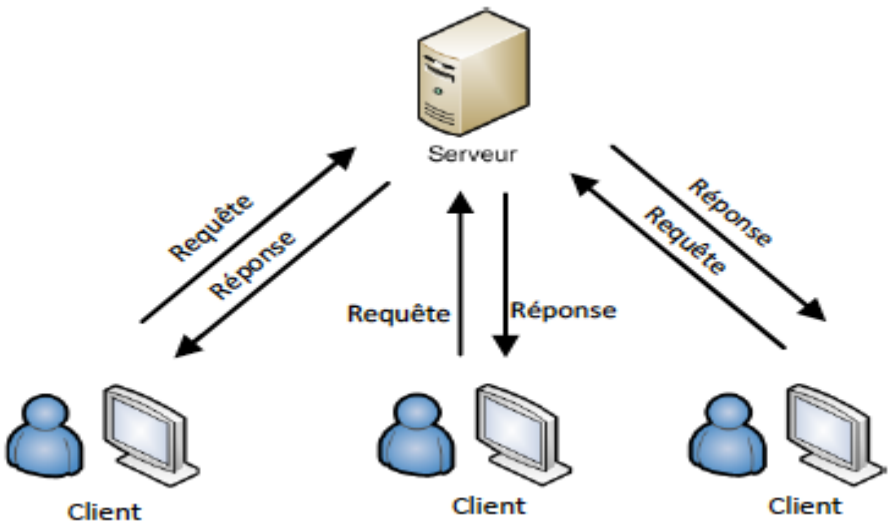

Figure 2. Un client/Un serveur.

Dans le modèle Client/Serveur, on distingue deux sous modèles selon que le service est effectué par un ou plusieurs serveurs (figure 2). 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).

Il existe diverses variantes du modèle Client/Serveur [9], 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.

3.2 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 traitements (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 traitements 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 3.

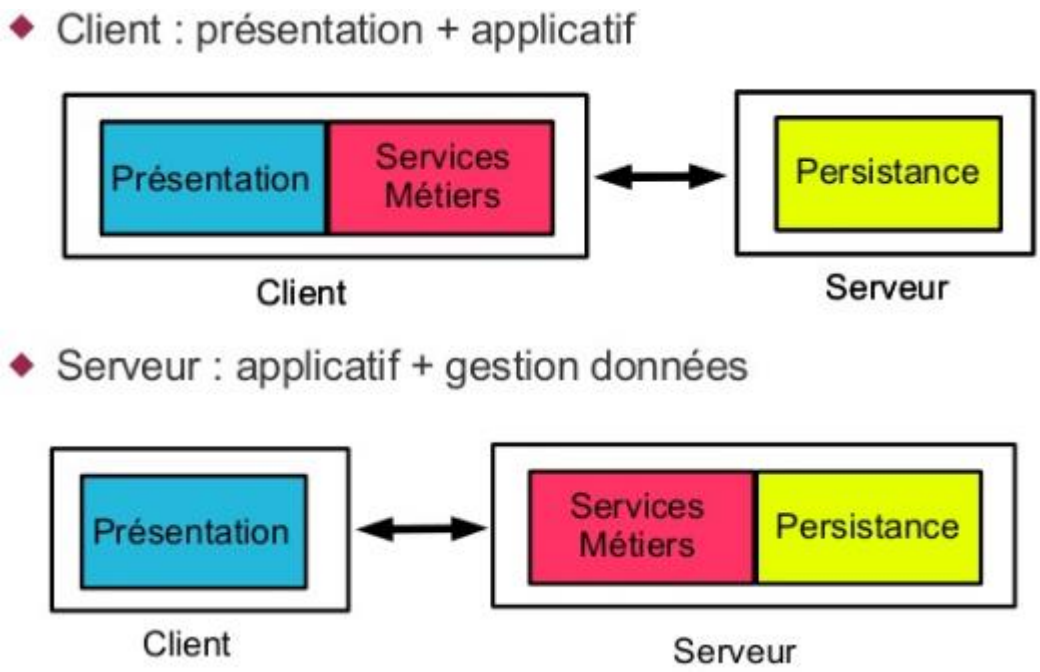

Figure 3. Architecture deux tiers

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.

Exemple :

Les applications Web sont des applications multitiers qui se caractérise par :

- Des clients identiques consistant en des navigateurs internet

- Des serveurs qui se composent principalement d'un serveur Web à l'écoute sur un port de communication et de programmes d'application avec éventuellement une base de données
- Des échanges entre les tiers qui respectent des normes et des protocoles standards

La programmation des applications Web consiste à concevoir et à écrire des programmes résidants dans un répertoire, défini comme étant un répertoire de scripts chez le serveur, et recevant leurs commande d'activation d'une page Web. Généralement cette page WEB utilise un formulaire HTML ou des liens directs pour lancer le programme ou le script. Le formulaire HTML est devenu le moyen le plus utilisé pour l'envoi des données sur internet, car il permet de mettre en place des fenêtres de saisie, des cases à cocher, des boutons radios, etc. ce qui facilite la récupération des données.

La démarche à suivre pour la conception d'une application web comporte les étapes suivantes :

- La création des pages Web contenants des références à des programmes exécutables, en utilisant la norme HTML.
- Les références sont généralement des liens directs d'activation des programmes (scripts) ou des boutons de soumission de formulaires qui seront remplis et à envoyés aux programmes de l'application.
- La création d'un ensemble de programmes qui reçoivent leur commande d'activation par des références dans des pages Web. Ces programmes ont généralement une interface standard du type CGI ou ISAPI, et résident dans un répertoire définie comme étant un répertoire de scripts dans l'arborescence des répertoires du serveur (généralement cgi-bin, scripts, etc.).
    - Les programmes reçoivent en entrée des arguments dits variables d'environnement qu'ils traitent, et génèrent en sortie des en-têtes de réponses compréhensibles par le serveur HTTP. Dans le cas où un programme ne fournit rien en sortie, la réponse sera vide.
    - La création des programmes (scripts) se fait par un langage permettant la définition des interfaces (ISAPI, CGI,…) des scripts (Java, Perl, Delphi ou autres).
    - La création d'une telle application nécessite un certain niveau de programmation ainsi qu'une certaine manipulation des serveurs (installation, configuration, etc).

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

Dans ce type d'architecture, il n'existe pas de distinction, en termes 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.

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.

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.

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.

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 données 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 4 illustre cette architecture.

Pour télécharger un fichier, l'utilisateur soumet à l'aide du servent une requête comportant les mots clés pouvant apparaître dans le 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.

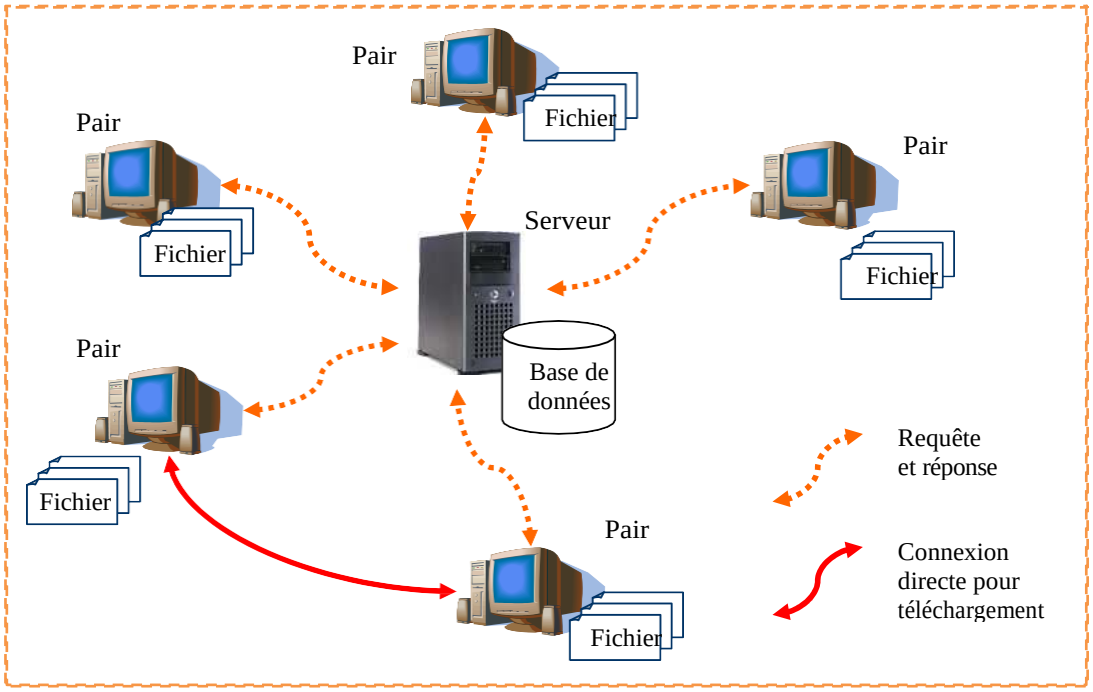

Figure 4. Architecture centralisée

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 5 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 préservent la bande passante). L'architecture en anneau est souvent utilisée lorsque les machines sont relativement proches (appartenant souvent à la même organisation).

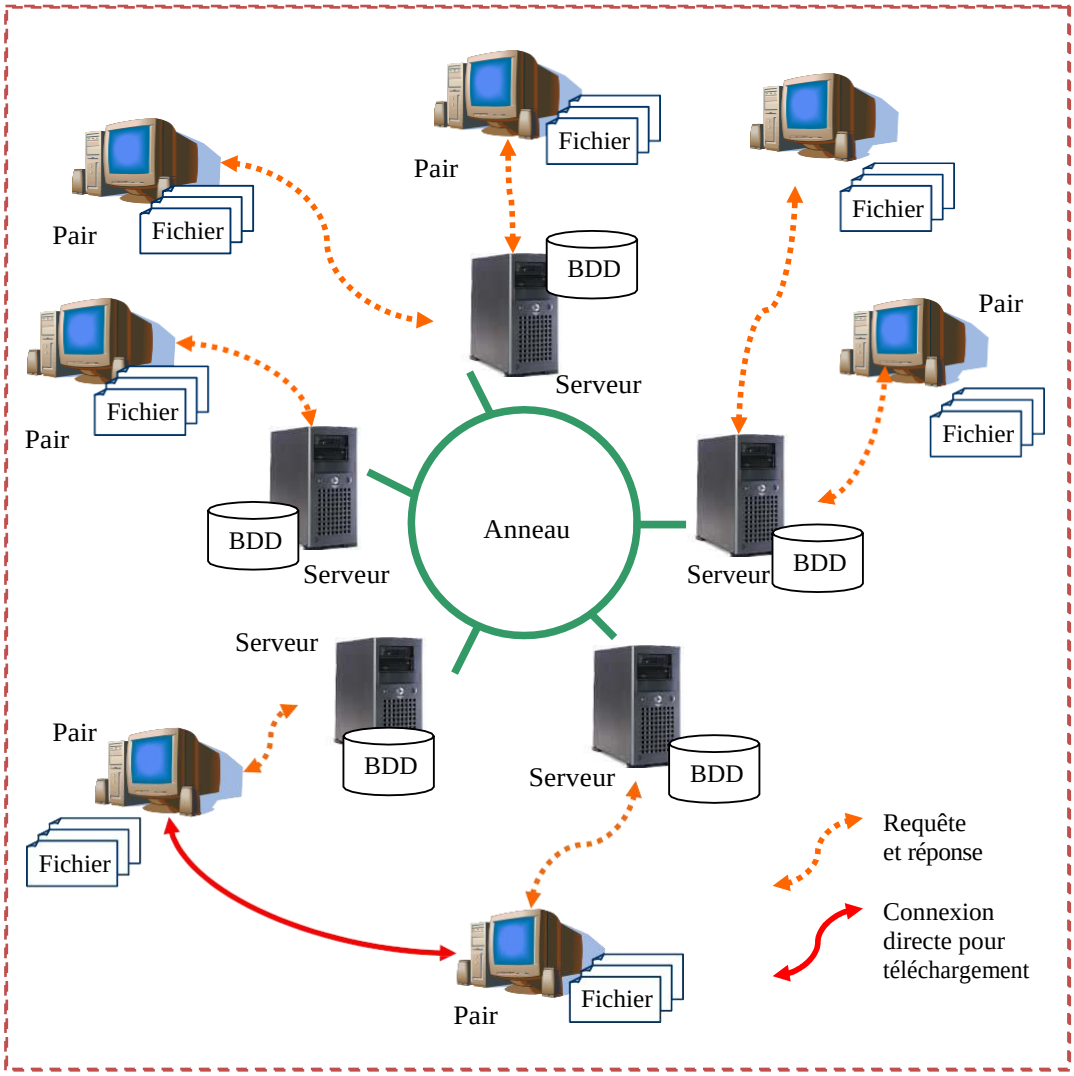

Figure 5. Architecture P2P en anneau

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 6 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 7 illustre l'approche.

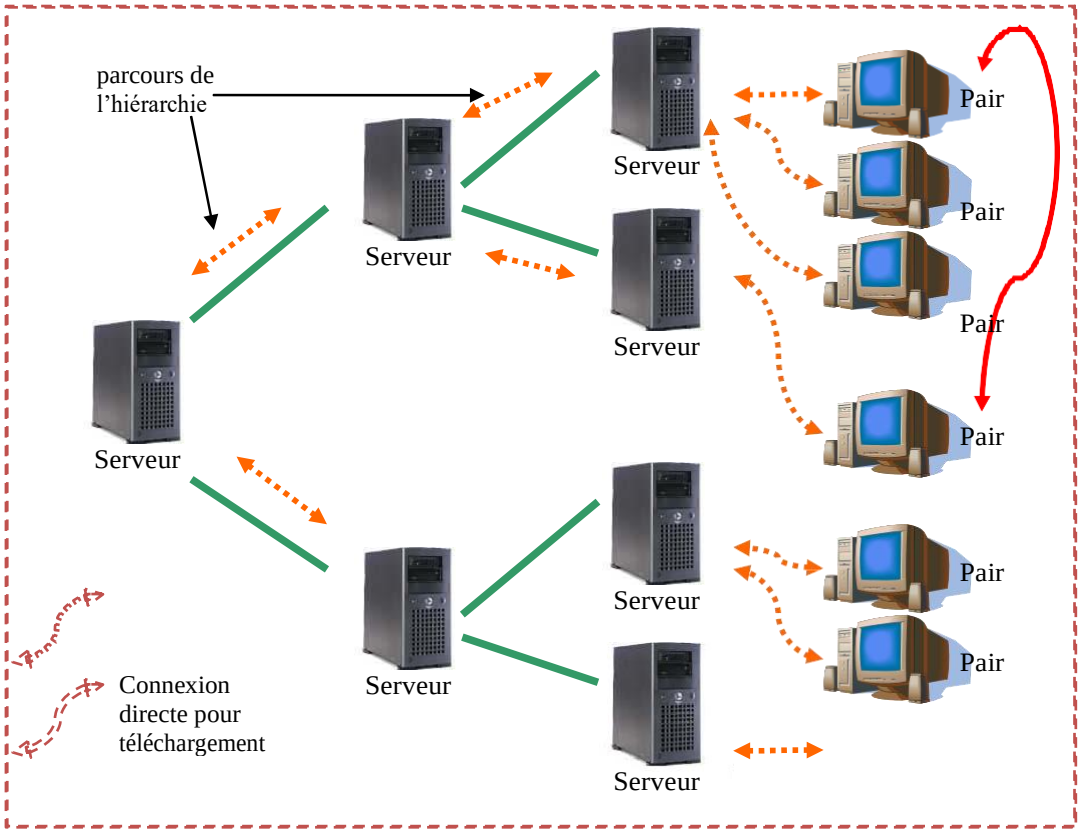

Figure 6. Architecture P2P hiérarchique

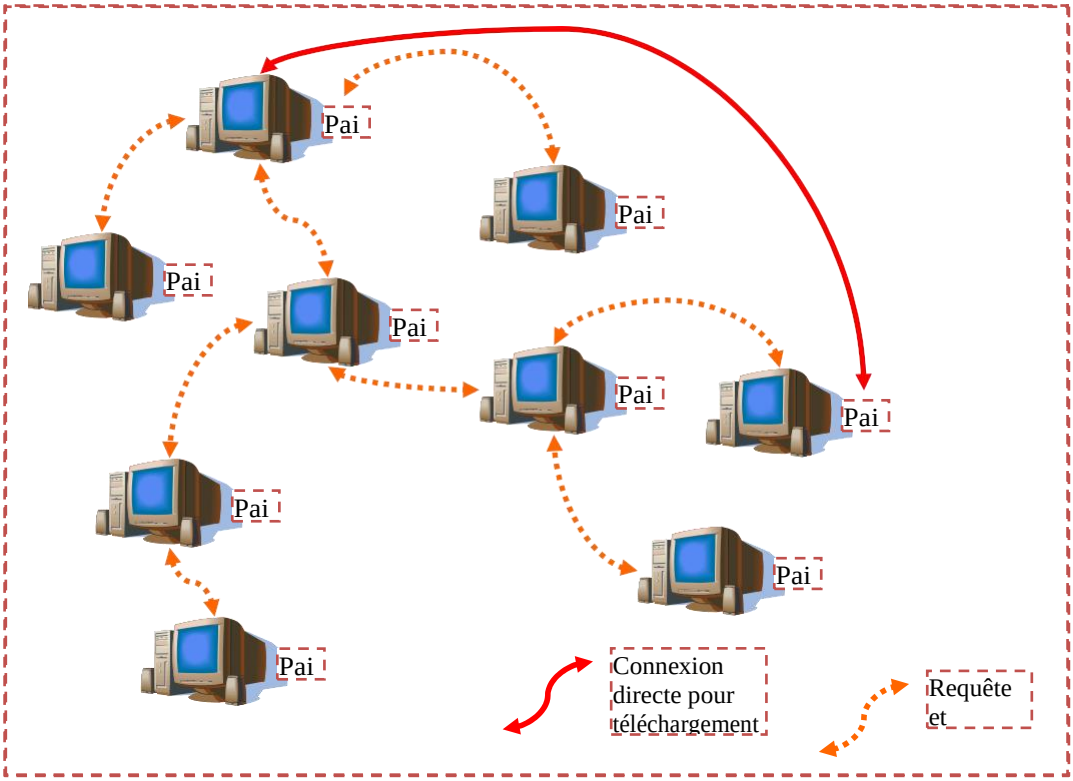

Figure 7. Architecture P2P décentralisée

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.

Référence :

[1] Nikolai Matni, Yoke Peng Leong, Yuh Shyang Wang, Seungil You, Matanya B. Horowitz, John C. Doyle 2014 Resilience in Large Scale Distributed Systems Procedia Computer Science 28 pp. 285-293. [2] Kovalev I., Zelenkov P., Ognerubov S. 2013 The minimization of inter-module interface for the achievement of reliability of multi-version software Proceedings of the 2013 International Conference on Systems, Control and Informatics (Venice, Italy) pp. 186- 189.
[3] Anton V. Uzunov, Eduardo B. Fernandez 2014 An extensible pattern-based library and taxonomy of security threats for distributed systems Computer Standards & Interfaces Volume 36, Issue 4 pp. 734-747.
[4] Tariq Omari, Greg Franks, Murray Woodside, Amy Pan 2007 Efficient performance models for layered server systems with replicated servers and parallel behavior Journal of Systems and Software Volume 80, Issue 4 pp. 510-527.
[5] Berrade M.D., Scarf P.A., Cavalcante C.A.V., Dwight R.A. 2013 Imperfect inspection and replacement of a system with a defective state A cost and reliability analysis, Reliability Engineering & System Safety Volume 120 pp. 80-87.

[6] Kovalev I.V., Polyanskiy K.V., Zelenkov P.V., Brezitskaya V.V., Sidorova G.A. 2013 System of search, analysis and processing of multilinguistic texts, integrated with data retrieval systems Vestnik SibGAU 1(47) pp. 48-52.
[7] Raspopin N.A., Karasyova M.V., Zelenkov P.V., Kayukov E.V., Kovalev I.V. 2012 Models and methods of data collecting and processing Vestnik SibGAU 2 pp. 69-72.
[8] Zelenkov P.V., Kovalev I.V., Karaseva M.V., Rogov S.V. 2008 Multilingual model of the distributed system on the thesaurus basis Vestnik SibGAU 1 pp. 26-28.
[9] Djamel Meslati, système distribues : principes et concepts, rapport pédagogique, 2010.