LE CLIENT - SERVEUR

1. Introduction

Au début des années 90, le nom de client/serveur n'était qu'une réalité pour quelques rares entreprises à la pointe de l'innovation et quelques fournisseurs capables de briser les frontières étanches des systèmes exclusifs ("propriétaires").

Depuis, le nom est devenu familier au point d'être utilisé de façon générique pour illustrer n'importe quel système qui fait se parler des applications différentes.

Surexploitée au risque de perdre toute identité, l'architecture **client/serveur** est pourtant **l'évolution la plus fondamentale** depuis l'arrivée du micro-ordinateur.

D'abord une **philosophie de l'ouverture**, l'architecture client/serveur est l'aboutissement des différentes mutations qu'à connues l'informatique : elle personnifie la **nouvelle architecture des systèmes de demain**.

L'informatique dispersée

Le paysage informatique au début des années 70 était très facile à appréhender. Les ordinateurs étaient tous de même nature, installés dans des endroits bien délimités et manipulés par un personnel strictement informaticien. Il n'était pas question de communiquer directement avec "la salle informatique". C'est à peine si quelques terminaux implantés au dehors s'autorisaient quelques connexions directes.

L'introduction au milieu des années 1970 du microprocesseur (et des micro-ordinateurs) a complètement changé la vue posée sur le monde informatique. Pour la première fois il était possible de construire un ordinateur complet, suffisamment petit et peu cher pour être utilisé par un individu seul et suffisamment puissant pour être réellement efficace dans le travail de tous les jours.

Le virage de l'informatique centrale ne s'est pas fait sans heurts. L'entrée d'IBM dans le monde de la micro-informatique a réduit les réticences de nombreuses entreprises à investir dans cette technologie.

La décentralisation de la puissance de traitement est maintenant une réalité : sur les bureaux et dans les attachés-cases, les processeurs totalisent bien plus de puissance que les systèmes centraux de l'entreprise.

L'informatique déconcentrée

La première phase de l'investissement des entreprises dans l'informatique personnelle s'est traduite par une prolifération de micro-ordinateurs affectés à des tâches personnelles, dans le but de les automatiser et d'améliorer la productivité individuelle.

La connaissance et l'expérience liées à cette informatique personnelle restent isolées au sein de chaque individu.

La seconde phase consiste à continuer à bénéficier des avantages d'une répartition de la puissance informatique au sein des utilisateurs tout en autorisant ces derniers à travailler ensemble et en garantissant l'intégrité des informations maintenant réparties hors des sites informatiques.

Elle consiste non seulement à donner à chaque individu une autonomie en termes d'outils individuels mais aussi un lien avec les applications de l'entreprise.

Le meilleur des deux mondes…

L'architecture client/serveur peut être considéré comme le meilleur des deux mondes informatiques :

- les avantages de la répartition transparente de la puissance informatique au niveau des utilisateurs et
- l'intégrité de la gestion coordonnée d'un système d'information cohérent.

Les deux mondes dont nous parlons sont bien identifiés :

- le monde des mainframes, autour duquel gravitent les mini-ordinateurs, super-minis, et autres super-ordinateurs et
- le monde des PC (ordinateurs personnels autour duquel gravitent les micro-ordinateurs, les portables, les stations de travail.

Les personnes pragmatiques pourraient décrire l'architecture client/serveur comme évitant le pire des deux mondes :

- la prolifération de données et d'applicatioions incohérentes isolées sur des PC

- la rigidité et le manque de convivialité des mainframes.

L'architecture client/serveur apparaît comme le dénominateur commun autour duquel l'ensemble du monde informatique se décline aujourd'hui.

L'architecture client/serveur semble apporter enfin un équilibre dans la prise en compte des intérêts des individus et de l'entreprise.

Les réseaux clé de voûte de l'architecture client/serveur.

Les réseaux de communication et les protocoles d'échanges constituent la clé de voûte de l'architecture client/serveur.

L'émergence des réseaux locaux d'entreprise n'est d'ailleurs pas étrangère au succès du concept client/serveur.

Le problème des normes est devenu central : le respect du standard OSI de l'ISO est probablement incontournable mais cela n'est pas une garantie de connectivité entre des systèmes ou des constructeurs différents.

Les protocoles OSI, avec d'autres standards de fait tels TCP/IP, IEE 802.3, constituent une base d'accord entre les constructeurs mais l'évolution très rapide des réseaux fait que le paysage se modifie très rapidement.

2. Le cadre théorique

2.1. Vers le modèle client-serveur

Le besoin de partage d'informations stockées sur des machines distantes met en jeu plusieurs stratégies de réalisation.

L'architecture client/serveur s'impose de plus en plus comme l'architecture de support des systèmes d'information car d'une part l'approche conventionnelle centralisée multi-utilisateur a montré ses limites face aux attentes des entreprises, et d'autre part plusieurs facteurs (technologiques et économiques) favorisent des solutions alternatives.

2.2. Le modèle centralisé multi-utilisateur

Le modèle traditionnel de l'architecture support des systèmes d'informations se définit comme un modèle centralisé multi-utilisateurs.

La centralisation indique que les données et les traitements sont supportés par une unique machine centrale, dont la taille (ou la puissance) est directement proportionnelle au volume de données traités et au nombre d'utilisateurs impliqués.

La première réaction des entreprises fut de délocaliser leur puissance informatique dans les départements spécifiques. Néanmoins nous sommes toujours dans un modèle centralisé multi-utilisateur : globalement le problème a été découpé (répartition de la charge de traitement sur plusieurs machines), mais chaque machine concentre un nombre d'utilisateurs travaillant sur leurs propres données.

2.3. Le modèle réseau local

D'abord un outil de productivité individuelle, le micro-ordinateur est devenu un outil de productivité de groupe : l'apparition de logiciels "réseau" appelés Groupware (Collecticiel selon AFNOR ou Synergiciel) a rappelé que la productivité globale dépend de la qualité des échanges entre les employés d'une même entreprise.

Ainsi, les réseaux locaux ont-ils commencé à proliférer, permettant de "mieux communiquer" : transfert de fichiers d'un ordinateur à un autre, transfert de messages, ...

On a ensuite assisté à une décentralisation plus ou moins anarchique de la responsabilité du développement d'applications et d'administration des données: certains micro-ordinateurs puissants ont été transformés en serveurs de fichiers mettant les données à disposition d'un groupe d'utilisateurs.

Une architecture reposant sur des micro-ordinateurs et des serveurs de fichiers en réseaux pose de nombreux problèmes: faibles performances des réseaux, difficulté de récupération après panne, difficultés de programmation, inconsistance et incohérence des données.

2.4. Faible performance des réseaux locaux

La plupart des PC reliés en réseaux locaux d'entreprise sont destinés à l'amélioration de la production individuelle, utilisant des applications de type traitement de texte ou tableur qui génèrent peu de trafic sur le réseau.

Le trafic existe uniquement lorsqu'un utilisateur lance un programme (lorsqu'il est chargé à partir d'un serveur de programmes) ou lorsqu'il sauvegarde ses fichiers sur un serveur de fichier. La plupart du temps, le traitement est fourni par la station "cliente".

Dés que l'on utilise une application de gestion de données, les conditions changent radicalement.

Prenons l'exemple d'une station consistant à produire un rapport historique sur l'ensemble des clients de l'entreprise à partir d'une requête du type "lister les clients de la zone B ayant réglé leur facture au premier trimestre".

Le programme et les données résident sur un serveur de fichiers mais le programme s'exécute entièrement sur la station cliente : après avoir transmis tout le programme sur la station, le serveur doit envoyer le fichier client en entier, enregistrement par enregistrement ! Les calculs sont effectués sur la station cliente qui cumule certains paramètres à partir de chaque enregistrement.

Une telle opération peut durer plus d'une heure si le fichier a transmettre est trop gros. Tous les autres utilisateurs actifs verront leurs performances se dégrader nettement durant ce transfert.

Cette architecture limite donc le nombre d'utilisateurs susceptibles de travailler simultanément de manière satisfaisante.

2.5. Difficulté de gestion des pannes

Par ailleurs les outils mis en oeuvre n'assurent pas toujours un niveau idéal de sécurité et d'intégrité des données.

Lorsqu'un ordinateur tombe en panne en plein traitement, il laisse généralement les données qu'il traite dans un état instable. Un "plantage" pendant l'exécution d'un programme comptable par exemple conduira à des balances incorrectes.

En particulier, une transaction fonctionnelle est souvent composée de plusieurs transactions unitaires (débit puis crédit de deux comptes). Les transactions unitaires sont effectuées généralement sur la station cliente puis les résultats sont envoyés sur le serveur de fichiers.

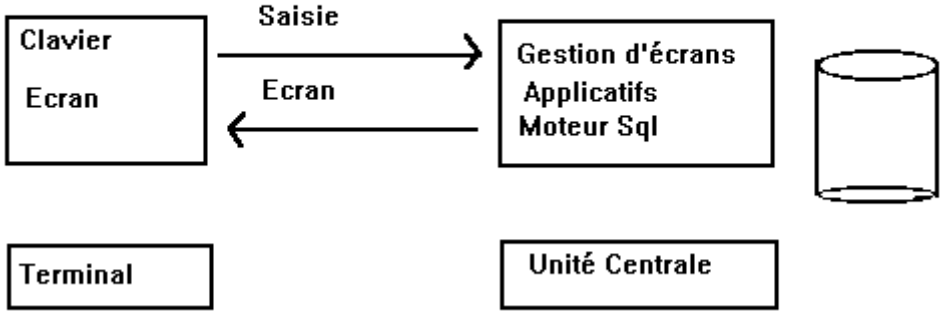
Si le processus est interrompu, la transaction unitaire est perdue, et la transaction fonctionnelle est incomplète, générant des données/résultats inconsistants.

Si on multiplie ce genre de problème par le nombre d'utilisateurs disséminés sur le réseau, remettre de l'ordre dans un système après une panne générale peut devenir un problème majeur pour l'entreprise.

2.6. Synthèse

Le système d'information central sur minis et grands systèmes maintient une cohérence satisfaisante des données de base pour l'utilisation d'applications de production, mais ne peut pas répondre aux besoins d'applications spécifiques de pilotage et d'analyse stratégique.

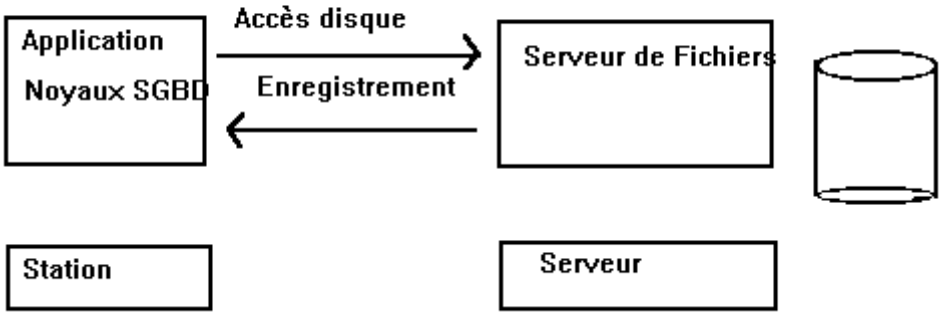
L'architecture Multiposte



- sécurité, fiabilité, intégrité
- grand volume de données

Les micro-ordinateurs reliés en réseau autour de serveurs de fichiers répondent à un premier niveau de besoins individuels ou de travail de groupe mais ne fournissent pas une bonne réponse aux besoins de l'informatique de production d'une entreprise.

L'architecture Réseau Local



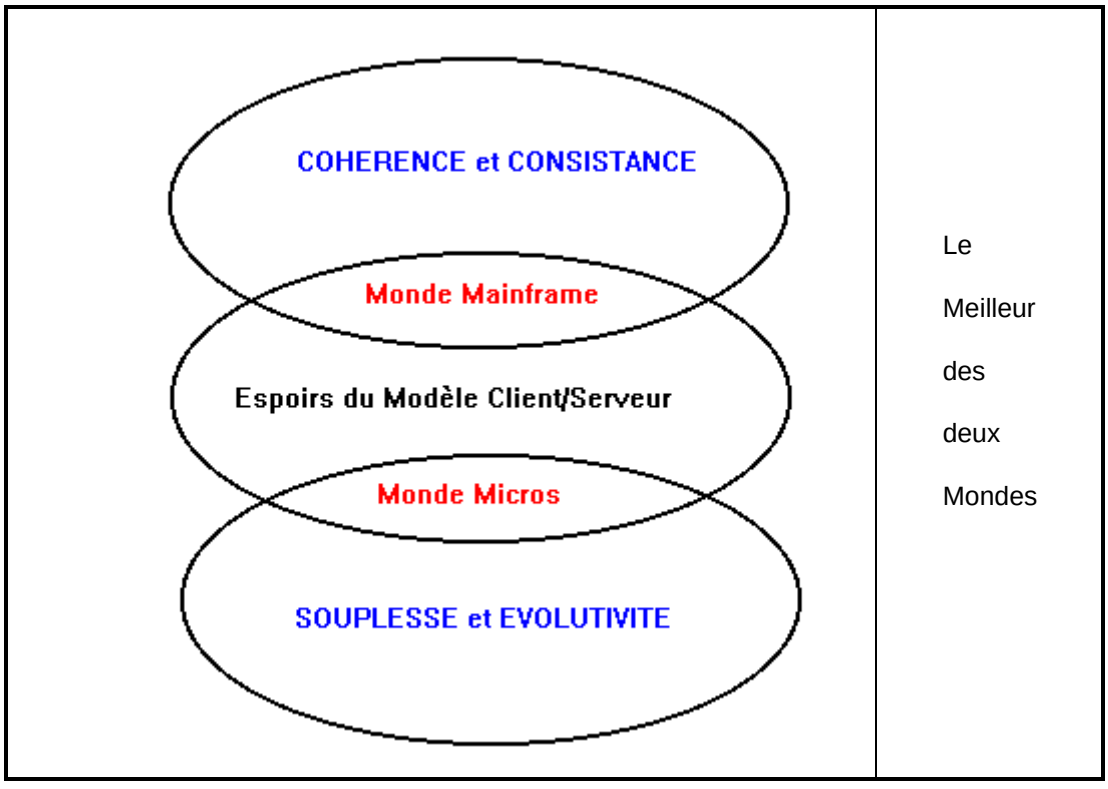
- interactivité, outils graphiques et individuels
- gros trafic réseau (toutes les opérations d'Entrées/Sorties), faible intégrité
- peu performant avec fort trafic

C'est entre ces deux situations que se situent les promesses de l'architecture Client/Serveur : maintenir la solidité et la consistance des données de l'entreprise, tout en répondant aux besoins évolutifs de ses différents départements.

3. Le modèle client-serveur

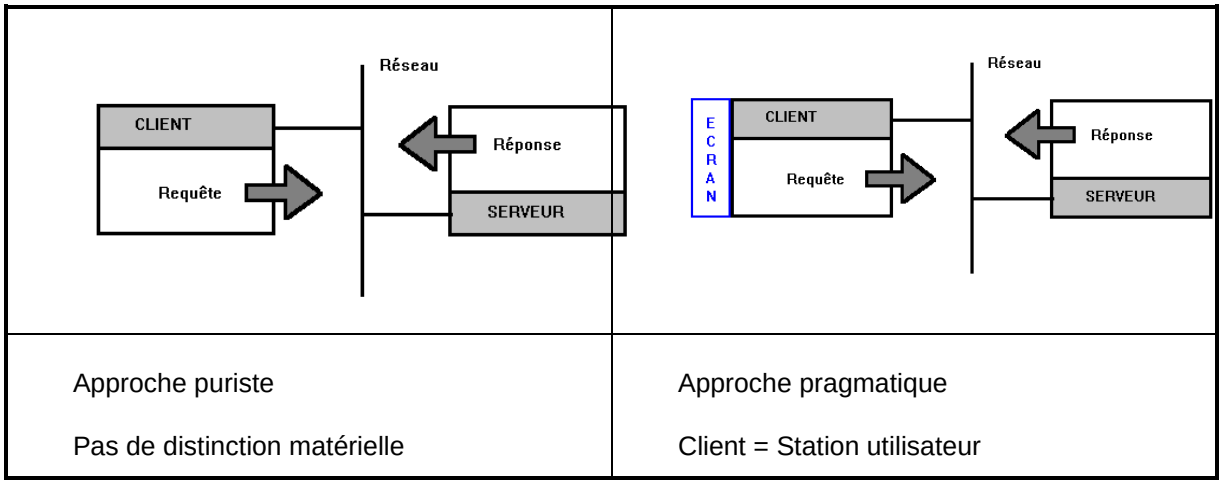
Les problèmes évoqués sont plus ou moins directement liés à cette notion de "vide" entre une architecture centralisée multi-utilisateur et une architecture décentralisée de micro-ordinateurs connectés.

Les fondements théoriques de l'architecture client/serveur permettent d'envisager une solution d'ensemble qui s'illustre bien par la métaphore du **meilleur des deux mondes**.



3.1. Le modèle théorique

Le modèle Client/Serveur est de façon théorique une architecture permettant le traitement coopératif d'applications, c'est-à-dire la communication directe de deux applications via un réseau, ou encore la communication directe entre deux processus d'une même application répartie sur un réseau.



Approche puriste	Approche pragmatique
Pas de distinction matérielle	Client = Station utilisateur

Le terme **SERVEUR** fait référence à tout processus qui reçoit une demande de service venant d'un client via un réseau, traite cette demande et renvoi le résultat au demandeur (le **CLIENT**).

3.2. Quatre principes de base

3.2.1. Principe 1

Rendre l'architecture matérielle transparente vis-à-vis des développeurs et surtout des utilisateurs finaux.

Pour mettre en oeuvre ce principe, un système client/serveur doit être réalisé autour d'un **réseau de haut niveau**, gérant les communications entre les différentes stations de manière intelligente (les stations communiquent entre elles au niveau logique, le réseau se chargeant de connaître la correspondance physique à ces noeuds logiques).

3.2.2. Principe 2

Rendre le niveau physique (et logique dans une moindre mesure) des bases de données transparent pour les développeurs (et les utilisateurs)

Le principe de découplage, généralement employé, consiste à utiliser un langage de requête standardisé (SQL par exemple) et de soumettre cette requête au réseau qui se chargera de la transmettre au(x) serveur(s) concerné(s).

3.2.3. Principe 3

Utiliser au niveau de chaque station (cliente ou serveur) l'ensemble matériel/logiciel le plus adapté

Contrairement à l'approche centralisée qui utilise des ordinateurs universels, l'architecture client/serveur permet de dédier différentes machines à des activités précises.

Dans un mode distribué, les entreprises ont commencé à appliquer ce principe de machines dédiées, en déchargeant la machine principale des travaux particuliers (utilisation de machines à tolérances de pannes pour des applications critiques, utilisation de super-ordinateurs pour des applications scientifiques.

La communication entre ces différents systèmes est souvent difficile, voire impossible: c'est souvent une raison suffisante, aux yeux d'une direction informatique, pour ne pas distribuer une application sur une machine spécialisée.

Le concept client/serveur permet d'utiliser des machines adaptées à des besoins précis, tout en sauvegardant la cohérence de l'ensemble.

Cette possibilité de pouvoir adapter chaque station, sans mettre en cause la cohérence d'ensemble a trois impacts importants :

- d'un point de vue technique, cela permet d'optimiser l'outil informatique en choisissant des matériels répondant à des besoins précis

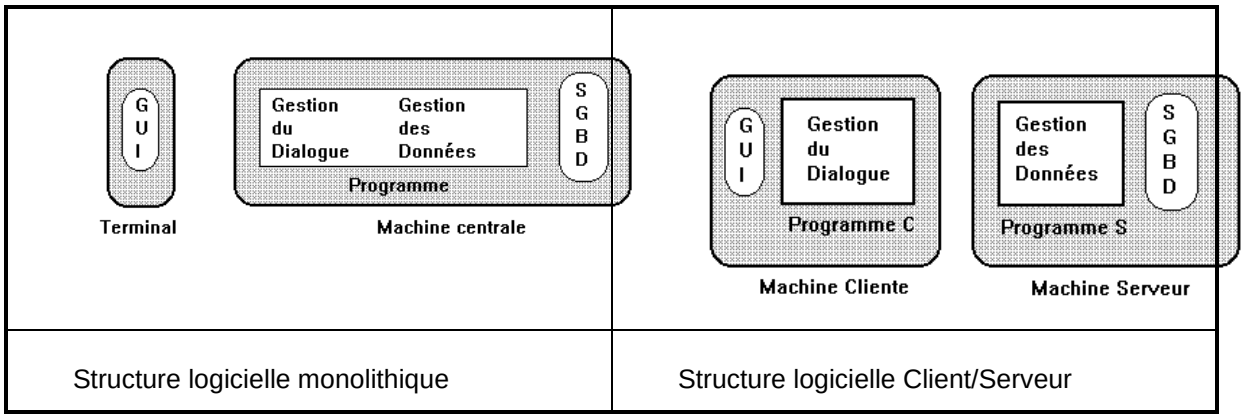
- d'un point de vue fonctionnel, cela permet d'offrir une diversité de services aux utilisateurs, correspondant à la diversité des besoins de chaque individu au sein de l'entreprise
- d'un point de vue économique, cela permet de minimiser les coûts en ne fournissant des machines sophistiquées que là où cela est nécessaire.

L'architecture client/serveur permet de répondre à un besoin d'intégration de matériels de constructeurs différents.

3.2.4. Principe 4

Permettre une séparation physique entre les actions d'un programme liées à l'interaction avec les utilisateurs et les autres actions.

Cette structuration des applications a déjà été prise en compte dans certaines méthodes de développement (développement de l'interaction avec l'utilisateur sous la forme d'une maquette, puis réalisation des autres fonctions).



Structure logicielle monolithique | Structure logicielle Client/Serveur

Ce principe de répartition du code d'un programme entre plusieurs machines sous-entend bien sûr un élément technique fondamental lié au modèle client/serveur :

c'est le support du traitement coopératif

(ou encore traitement distribué) ou communication de programmes à programmes.

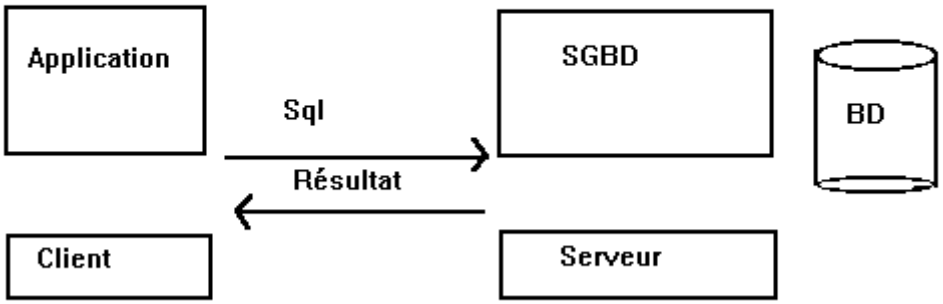
L'exemple des serveurs de bases de données

Contrairement aux serveur de fichiers, les serveurs de base de données respectent les concepts de l'architecture client/serveur :

- architecture distribuée,
- optimisation de l'utilisation des matériels,
- augmentation du niveau de transparence,
- traitement réparti
- structure des applications en deux parties.

Un système serveur de bases de données répond à chacun des problèmes évoqués sur l'insuffisance des réseaux locaux et des serveurs de fichiers.

L'architecture Client/Serveur de bases de données



Le modèle client serveur à pour objectif de répartir les tâches entre les systèmes, et par là même de réduire le trafic sur le réseau.

Le terminal client, généralement un PC, un Macintosh ou une station graphique assure la gestion d'une interface utilisateur, le traitement local des informations et l'adressage des requêtes SQL au serveur.

Le serveur, quant à lui, est responsable de la gestion de la base de données en termes de sécurité et d'intégrité, ainsi que de l'expédition des réponses aux différentes demandes formulées par ses clients.

La grande différence par rapport à une architecture de serveurs de fichiers classique réside dans le fait que ce serveur n'envoie au client que les données correspondant à sa requête limitant ainsi le débit sur le réseau et l'occupation du poste client.

En fait, le modèle client-serveur sous-tend le concept **d'architecture distribuée**.

4. Vers des architectures distribuées

4.1. Une hiérarchie de concepts

Tour à tour Downsizing, Interopérabilité, Systèmes Ouverts, Modèle Client/Serveur, Traitement distribués sont utilisés. Ces termes ne sont pas interchangeables, au contraire ils abordent chacun un aspect fondamental de l'évolution des architectures informatiques vers plus de flexibilité et de transparence.

Si on classe ces termes par ordre de complexité, le downsizing reste le plus facile à aborder.

4.1.1. Downsizing : concept uniquement économique.

Cette politique de réduction ou de l'ajustement de la taille des équipements informatiques est une référence économique faisant appels aux coûts d'achat et d'exploitation des différents types de plates-formes. Le coût du MIPS est un indicateur sans équivoque (MIPS sur PC inférieur à 100 fois celui des mainframes).

On reconnaît là un principe de l'architecture client/serveur : pouvoir modifier les éléments matériels sans qu'il faille revoir tout le système.

4.1.2. Ensuite vient la définition de l'interopérabilité,

c'est à dire le concept de systèmes ouverts qui repose en grande partie sur l'adoption de standards. Pour qu'une architecture puisse être considérée comme ouverte, il faut que les différents éléments qui la composent aient des caractéristiques de standardisation et de portabilité que des organismes internationaux mettent sur pied :

- le respect des modèles standards pour la communication,
- une base commune pour les systèmes d'exploitation,
- l'existence d'interfaces de programmation normalisées pour les accès entre systèmes hétérogènes.

De fait l'interopérabilité comprend trois grand volets que sont :

- la CONNECTIVITE dont le modèle d'interconnexion OSI est la référence,
- la PORTABILITE qui se décline autour du monde Unix, mais dont POSIX (Portable Operating System Interface) sera la base,

- l'INTEROPERABILITE des OPERATIONS qui va regrouper les questions tenant aux API, à la normalisation SQL, ...

4.1.3. L'architecture client/serveur

proprement dite se présente sous un abord plus technique : il s'agit d'interfaces de programmation (API), d'interfaces de communication (PCI), et d'appels de procédures distantes (RPC)

4.1.4. Enfin, la distribution,

qu'il faut voir à la fois du point des traitements et des données. La théorie de la distribution doit permettre de gérer un ensemble de ressources logiques, physiquement dispersées que l'on peut rassembler et re-disperser à volonté.

Au lieu d'avoir accès à un unique serveur de données, un poste client peut accéder de manière transparente à plusieurs serveurs.

Cela signifie qu'une même requête peut concerner des données réparties sur plusieurs systèmes , mais que l'émetteur de la requête n'a pas à se soucier de la localisation et de l'implantation des données. Ces traitements complexes sont pris en charge par des SGBDR répartis, capables pour certains de gérer un environnement hétérogène de matériels et logiciels.

5. Vers le traitement coopératif

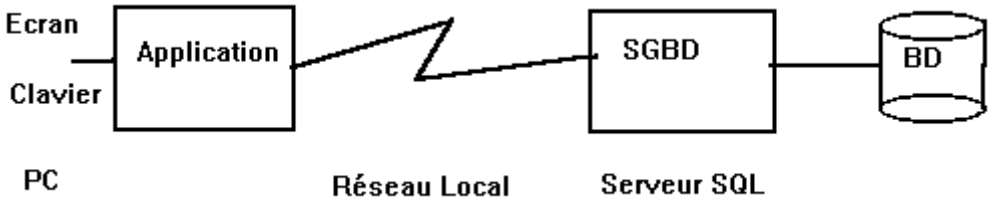
Le traitement coopératif nécessite la participation (coopération) de deux composants situés sur des équipements différents.

Le modèle Client - Serveur est un exemple de traitement coopératif. En effet cette architecture suppose l'existence de deux objets :

- Client : programme demandant l'accès à un service ou à une ressource.
- Serveur : programme rendant le service ou attribuant la ressource.

Ces "objets" communiquent au travers d'interfaces (standards ou spécifiques) dans le même système ou dans deux systèmes différents.

Exemple :



Les caractéristiques de ce modèle sont :

- architecture purement logicielle
- indépendance des deux composants
- distribution possible sur des systèmes différents
- partage d'un serveur entre plusieurs clients
- accès par un même client à plusieurs serveurs

Ce modèle est un "modèle" pour le développement d'applications distribuées.

Les domaines d'utilisation sont :

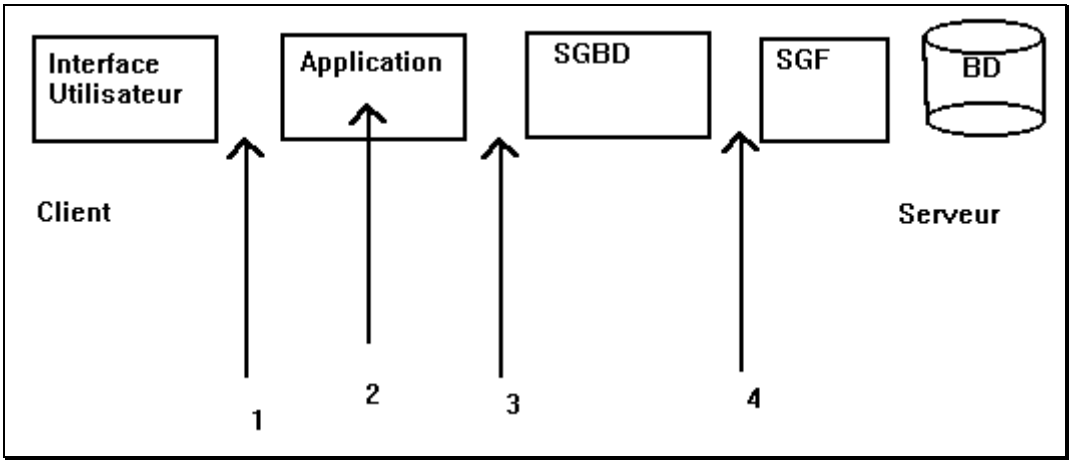
- le partage de fichiers (sur des réseaux locaux)
- l'accès à des bases de données
- le partage de processus de traitements spécialisés
- les interfaces homme-machine

La mise en place d'une application sur 2 systèmes introduit des problèmes nouveaux.

Le choix de la distribution :

- Quelles procédures seront clientes ?
- Quelles procédures seront serveurs ?

Exemple :



La difficulté consiste à choisir le niveau de 1 à 4 pour séparer le client du serveur.

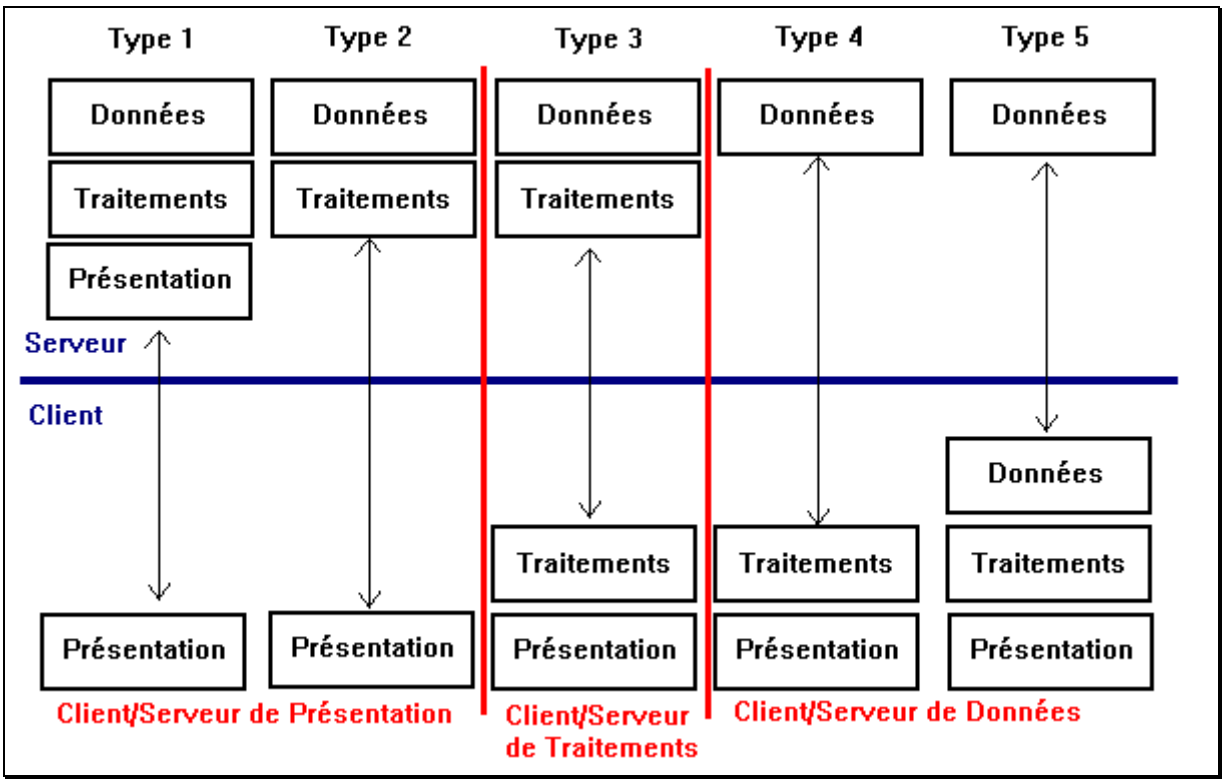
La répartition doit tenir compte :

- des types d'architectures retenues,
- de la capacité des machines,
- des capacités du réseau

Les différents types de Client/Serveur

6. Schéma du Gartner Group

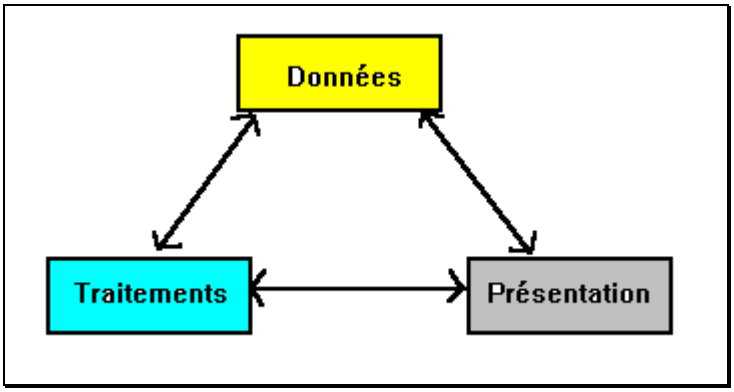
Le Gartner Group est une société américaine de consultants qui a publié un schéma des différents types de client/serveur.



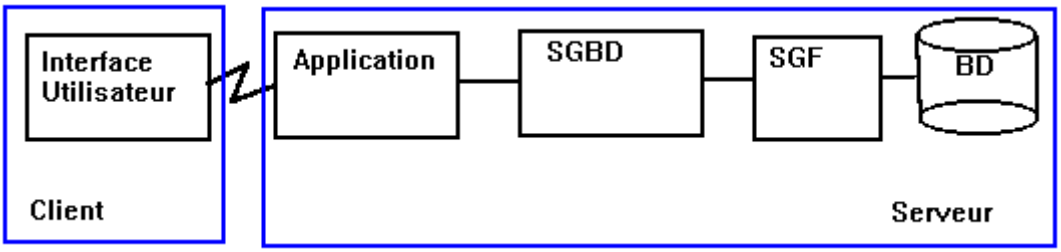
6.1. Type 1

La décomposition d'une application est résumée en trois modules :

- Présentation : tout ce qui concerne l'interface avec les utilisateurs
- Traitements : logique applicative
- Données : accès aux données gérées par un SGBD

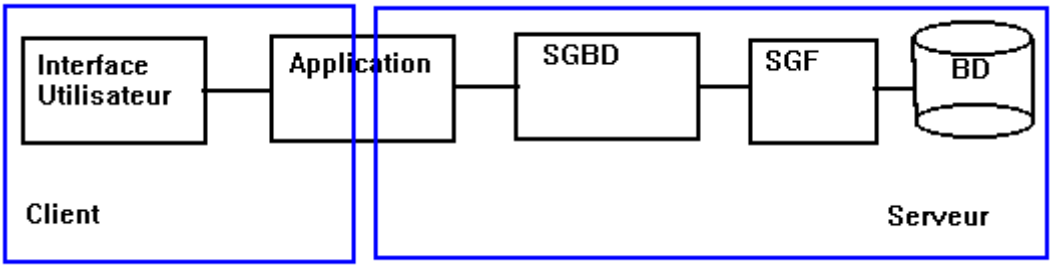


6.2. Scénario Type 2 : Présentation déportée ou distante



La mise en oeuvre du client/serveur de présentation suppose de pouvoir séparer la gestion de l'affichage de la logique d'affichage

6.3. Scénario Type 3 : Logique d'application distribuée



Ce type est également qualifié de traitement distribué ou client-serveur de procédure.

Avec le client serveur de procédures, on accentue un peu la difficulté car il ne s'agit plus de déporter les services extrêmes de l'application (tout en haut l'interface ou tout en bas les données) mais bien de procéder à un découpage au plus près du noyau de cette application.

Il requiert plus de savoir-faire pour sa mise en oeuvre, ce qui explique que cette forme de répartition théoriquement la plus efficace est loin d'être la plus répandue.

Le client effectue la gestion du dialogue, la validation des saisies et la mise en forme des résultats et la partie traitement de l'application.

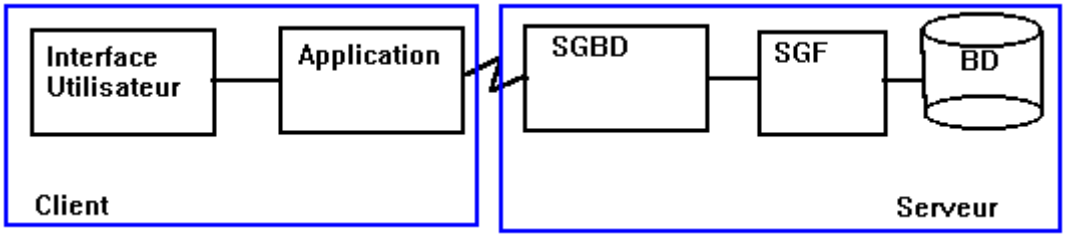
Le serveur effectue la partie manipulation des données et gère l'accès aux données.

Les SGBD Relationnels de génération récente permettent l'utilisation des procédures stockées.

Dans ce cas, au lieu d'envoyer des requêtes SQL (comme c'est souvent le cas du client/serveur de données), l'applicatif client envoie des appels de procédures qui elles; contiennent une ou plusieurs requêtes. La structuration du code stocké dans la base de données diffère d'un SGBD à l'autre. Avec **SQLServer** de Microsoft, vous utilisez le

langage *TransactSQL* et avec **Oracle** vous devez utiliser le langage *PL/SQL* pour créer des procédures ou des fonctions.

6.4. Scénario Type 4 : Gestion déportée ou distante des données



Le client/serveur de données est l'exemple le plus répandu de mise en oeuvre du client/serveur.

Dans ce cas les choses sont claires :

Le client effectue la gestion du dialogue; la validation des saisies et la mise en forme des résultats et les traitements applicatifs (y compris la manipulation des données)

Le serveur gère l'accès aux données et de plus en plus souvent l'intégrité des données. Il est appelé serveur de données.

Cette mise en oeuvre est très répandue car elle a été popularisée par les SGBD relationnels qui ont reposé dés leur origine sur le modèle client/serveur pour ce qui est de leur fonctionnement réparti. Par la suite, quasiment tous les éditeurs ont commercialisé un IPC (Inter Process Communication) permettant d'accéder à leur produit à travers un réseau depuis une station Dos, puis Windows. Oracle à commercialisé SQL*Net qui est le nom de son IPC propriétaire (ou produit Middleware) alors que Microsoft a mis en avant son concept d'ODBC (Open DataBase Connectivity).

Exemple : Application SQL : VisualBasic avec SQLServer 6.5

7. La distribution des données

Outre l'accés aux données via ODBC (client-serveur de données) une architecture client/serveur implique trés souvent une distribution des données sur d'autres plateformes SQLServer ou non (ORACLE, SYBASE, DB2, ACCESS,...)

7.1. L'import-export

Tout SGDB serveur offre la possibilité d'importer et d'exporter des données.

SQL Sserver permet ces échanges en mode natif SQLServer ou en mode caractère (ASCII).Ceci est fait par un utilitaire appelé **BCP** pour *bulk Copy Program*.

BCP peut aussi permettre de récupérer des données issues de plusieurs tables pour les envoyer vers un tableur ou un SGBD de bas niveau.

7.2. La réplication

La réplication est un mécanisme asynchrone de duplication de données. L'idée est de rendre disponibles des données sur plusieurs serveurs, afin d'améliorer leur disponiblité et d'en faciliter l'accés.

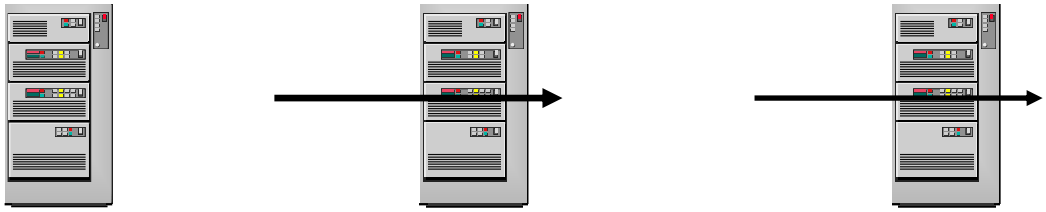
Asynchrone : la distribution est faite périodiquement (ex : tous les soirs) mais pas en temps réel.

Le vocabulaire employé est le suivant :

- Serveur de publication (*Editeur*)
- Serveur de distribution (*distributeur*)
- Serveur d'abonnement (*Abonné*)

L'éditeur contient les données à répliquer.

Le *distributeur* stocke temporairement les données d'un ou plusieurs éditeurs avant de les ditribuer vers les *abonnés*. L'abonné reçoit les infos répliquées.



Serveur de publication (Editeur) | Serveur de distribution (Distributeur) | Serveur d'abonnement (Abonné)

La réplication est un processus qui se décompose en 3 phases.

→ **La synchronisation** : La synchronisation fait référence à la mise en phase des structures et des données des tables répliquées entre l'éditeur et l'abonné. (Contenu à répliquer , fréquence).

Les scripts correspondants aux structures à répliquer sont contenus dans le répertoire.

/MSSQL/REPLDATA pour SQL Server.

→ **La lecture du journal** : Le processus de lecture du journal se déclenche à échéance fixée par l'administrateur.

La lecture du journal de transactions permet de rechercher dans la base publiée les transactions marquées pour la réplication. Elles sont appliquées à la base *distribution*.

Vous remarquez bien que seules les modifications apportées aux données publiées sont distribuées vers le client.

Ceci permet d'éviter d'envoyer la totalité des données contenues dans les tables à publier et donc de diminuer le trafic réseau.

→ La distribution :

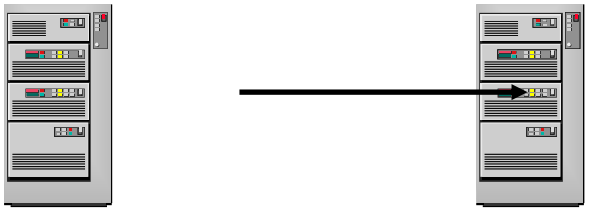
Une fois écrite dans la base de distribution, les transactions sont envoyées immédiatement vers les abonnés.

Dans SQL Server, les tâches de réplication sont assurées par **SQLExecutive** qui doit avoir un compte sous WINDOWS N.T.

Ce compte doit :

- faire partie du groupe Administrateurs
- avoir un mot de passe qui n'expire jamais
- ne pas avoir de limites d'heure d'ouverture
- avoir le droit d'ouvrir une session en tant que service

Trés souvent le même serveur joue le rôle d'éditeur et de distributeur



Serveur de publication et de distribution (Editeur et Distributeur) | Serveur d'abonnement (Abonné)

Que peut-on publier ?

L'unité de base de la publication est l'*article*

Un article peut être

- -une table
- -une projection sur une table
- -une sélection sur une table

Un ensemble d'articles forme une *publication*.

Mise en oeuvre :

Sur **SQLServer** la mise en oeuvre de la réplication est faite avec l'utilitaire **SQL Enterprise Manager** dans les menus *Serveur / Configuration de la réplication*.

Vous disposez des assistants qui vous permettent de

- -sélectionner le distributeur (en premier)
- -sélectionner l'éditeur
- -sélectionner les articles à publier

Exemple d'utilisation :

- Une centrale d'achats est chargée de l'approvisionnement pour plusieurs supermarchés.
- Un système de distribution des données a été mis en place entre la centrale et chaque supermarché.
- La fréquence de réplication est quotidienne (le soir à 23 heures).

Le plus important pour chaque supermarché est de connaitre les produits que propose la centrale, le coût d'achat, les promotions

Au niveau de la centrale, on dispose de tables

Tables de référence	Tables répliquées
FOURNITURES PRODUITS CDE-FOURNISSEURS	CDE-SUPERMARCHE DETAILS-CDES-SUPER

Au niveau de chaque supermarché, on aura des tables pour la gestion interne et la réplication se fera essentiellement sur la tables PRODUITS pour connaitre chaque jour le prix auquel le supermarché peut commander à la centrale.

On aura donc

Tables de référence	Tables répliquées
CAISSE VENTES CDE-SUPERMARCHE DETAILS-CDES-SUPER	PRODUITS

8. Les serveurs distants et les transactions distribuées.

Si on souhaite que les données soient immédiatement mises à jour (mode synchrone), la réplication ne convient pas.

Il faut s'orienter vers une solution d'informatique répartie.

Cela suppose l'utilisation de plusieurs serveurs qui seront synchronisés.On parle de validation à deux phases (**Commit à 2 phases**)

Exemple : soit 2 serveurs **S1** et **S2**

Un utilisateur souhaite faire une requête qui récupère les données sur tous les clients et prospects or les clients sont sur le serveur **S1** et les propects sur le serveur **S2** .

L'utilisateur va exécuter une procédure stockée qui existe sur le serveur S1 et qui contient une requête sur le serveur S2 ce qui peut être transparent pour l'utilisateur.

Ceci ne pose pas de problèmes lorsqu'il s'agit d'accés en lecture. Pour la mise à jour de données c'est autre chose.

Ex : **S1** est à paris et **S2** est à Madrid.

Un client fait un modification à Paris qui doit être repercutée sur le serveur à Madrid.

La procédure classique consiste à écrire une procédure stockée *MajParis* et une *MajMadrid* et à les exécuter sur leur serveur respectif en les « encapsulant » dans une transaction.

On a donc

BeginTrans

- Exec ServeurParis.BaseParis.dupont.MajParis
- Exec ServeurMadrid.BaseMadrid.dupont.MajMadrid

CommitTrans

Problème ! Si tout se passe bien à Madrid et que Paris échoue *MajMadrid* devrait être annulée mais le journal des transactions se trouve à Madrid et la transaction a été enregistrée à Paris ! Madrid ne voit que l'exécution de sa procédure . Paris ne peut pas annuler *MajMadrid* ce qui provoque une désynchronisation des contrôleurs.

Il faut donc un journal de transactions commun aux deux serveurs pour résoudre ce problème ! C'est en fait le rôle de **DTC** (**D**istributed **T**ransaction **C**oordinator) dans SQL Server.

Si le service **DTC** est en marche sur WINDOWS NT, il se charge de suivre les transactions sur plusieurs serveurs. **DTC** peut être démarré sur un seul des deux serveurs impliqués par la transaction.

Pour que la transaction précédente devienne une transaction distribuée, il faut remplacer

Begin Trans par Begin Distributed Trans

9. Internet – Intranet

Les données d'un serveur de base de données comme SQL Serveur peuvent être présentes dans des pages HTML et ceci de deux façons.

Microsoft propose 2 stratégies (ou modèles) de diffusion des données sur internet.

9.1. -le modèle *Push*

Dans ce modèle un assistant vous permet de créer des pages HTML et de les mettre à jour automatiquement ou manuellement. Un explorateur internet (navigateur, butineur) pourra alors permettre la consultation de ces pages.

C'est donc un modèle statique.

9.2. -Le modèle *Pull*

Ce modèle permet l'exécution de requêtes dynamiques.

Il est alors possible de visualiser des pages et d'accéder aux données du serveur à l'aide d'un navigateur. Il existe plusieurs produits permettant de faire ceci. Une 1ère solution est l'utilisation conjointe de Internet **I**nformation **S**erver

(**IIS**) et de Internet **D**atabase**C**onnector (**I.D.C**) et de **SQL Server**. Une autre solution apparue à l'automne 98 est l'utilisation de **Visual Interdev 6.0** et de **SQLServe**r.

Il est fort probable que de nombreux outils fassent leur apparition sur le marché pour permettre le traitement dynamique des données dans des pages HTML