Systèmes et applications répartis

Introduction : définition, contexte, exemples
- Caractéristiques d'un système réparti
- Conséquences
- Importance des systèmes répartis

Exemples
- Comment UNIX est devenu réparti
- WWW
- Intégration d'applications patrimoniales
- Télévision interactive
- Commerce électronique
- Accès à un fichier distant

Bilan : besoins induits par la conception d'applications réparties
- Prise en charge des problèmes spécifiques à la répartition → transparence
- Systèmes ouverts, à large échelle

Plan du cours

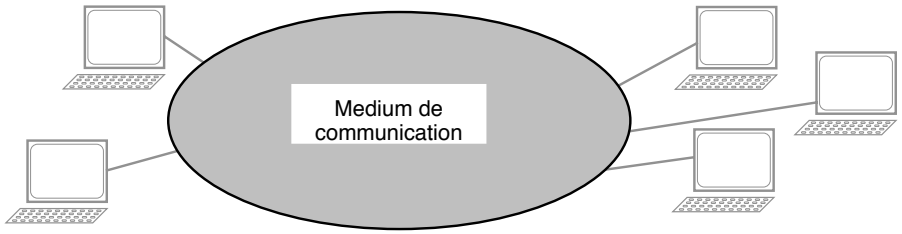
Crédits *: ce cours a été construit à partir des sources suivantes :*

- G Coulouris et coll : Distributed Systems, Addison-Wesley

Certains dessins et schémas de ce cours sont directement issus des cours de S. Krakowiak

1 – Introduction

Caractéristiques d'un système réparti :



ensemble de processeurs (sites) ***communiquant*** via un *réseau* ***asynchrone*** (délais de communication non bornés), tel que les différents sites sont :

- *indépendants* du point de vue de l'activité, mais
- *liés* par la réalisation ou l'utilisation d'un *service commun*, *global*

Difficulté de conception essentielle : incertitude irréductible sur l'état instantané du système
- synchronisation directe impossible : pas de mémoire ni d'horloge partagée entre sites
- synchronisation via la communication impossible : délais de communication non bornés, et vivacité de l'interaction non garantie en cas de panne/absence d'un correspondant
  → en particulier, la panne d'un site ne peut être détectée.

Services de base à fournir à l'utilisateur/au programmeur
- communication entre sites
  - ◊ mise en œuvre des interactions : acheminement des messages, coordination entre sites
  - ◊ gestion de l'hétérogénéité
- gestion/mise en œuvre des ressources partagées
  - ◊ concurrence
  - ◊ sécurité : confidentialité, intégrité, contrôle d'accès
- gestion de la dynamique du système
  - ◊ extensibilité, ouverture
  - ◊ prise en compte du facteur d'échelle
  - ◊ fonctionnement en mode dégradé, traitement des pannes, disponibilité
- localisation/désignation des sites, ressources, objets, activités
  → difficultés : évolution et taille de l'espace de noms, mobilité
- gestion de l'incertitude inhérente à la répartition : protocoles (vus plus tard)
  - ◊ de synchronisation et d'ordonnancement logiques/physiques,
  - ◊ de mise en cohérence d'informations répartie,
  - ◊ d'accord global (consensus)

Objectif
- mise en œuvre *transparente* des différents services de support à l'exécution répartie

Les systèmes répartis sont répandus...

Un grand nombre d'applications est de fait réparti
- parce que cela est « naturel »
  - ◊ composition de traitements concurrents distants et distincts
    → interaction et partage de données, de ressources
  - ◊ répartition et parallélisme intrinsèques des systèmes physiques
    → répartition intrinsèque des applications informatiques déployées sur ces systèmes
- parce que cela est possible
  - ◊ accroissement des performances, diminution des coûts } { processeurs, réseaux
  - ◊ couverture large (voire omniprésence) des moyens de calcul et de communication
  - ◊ interconnexion généralisée (interpénétration informatique-réseau-médias)

Applications réparties : un échantillon

- Coordination d'activités
  - ◊ systèmes à flots d'information (workflow)
  - ◊ agents mobiles
- Communication et partage d'informations
  - ◊ réseaux domestiques → intranets → W3
  - ◊ collecticiels (groupware), édition coopérative : wiki, gestionnaires de versions...
- Applications temps-réel
  - ◊ Systèmes embarqués (avionique → systèmes enfouis)
  - ◊ contrôle de procédés
- Multimédia
  - ◊ téléenseignement
  - ◊ visioconférence
  - ◊ télévision interactive
- Services commerciaux
  - ◊ gestion de stocks en temps-réel
  - ◊ systèmes bancaires
  - ◊ systèmes de réservation
  - ◊ commerce électronique
- Systèmes mobiles

2 – Exemple : comment UNIX est devenu réparti

Systèmes répartis ≠ systèmes (multiutilisateurs) centralisés (ex : UNIX originel)

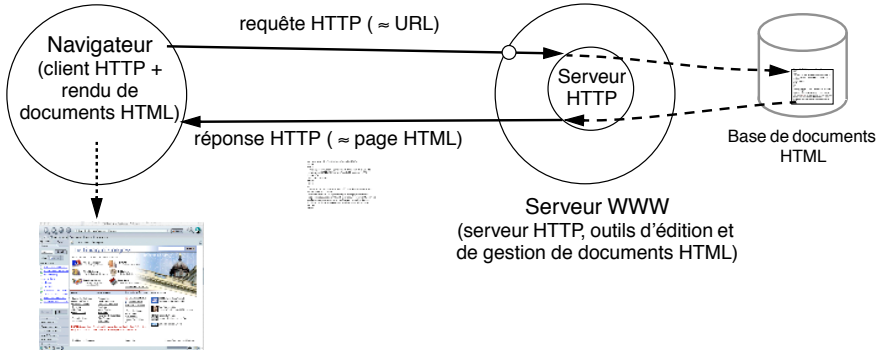
But : assurer et étendre { performances, fiabilité, extensibilité, communication } les fonctions des systèmes centralisés :

gestion { ressources matérielles, activités utilisateur }

Evolution

- définition de services de communication interprocessus (IPC) (~ 1978)
  → serveurs de ressources (fichiers, logiciels…) + processsus clients
- masquage/allègement de la communication
  RPC, NFS, NIS
- ajout de fonctions nouvelles
  Kerberos (sécurité), Andrew (fichiers répartis)
- accent sur l'ouverture (modularité, extensibilté)
  micro-noyaux : Mach, Chorus
  intergiciels : CORBA, .NET, EJB

3 – Exemple : WWW



Ingrédients

Protocole client/serveur

Evolution

Support d'accès à un système d'information (hypermédia) partagé et réparti

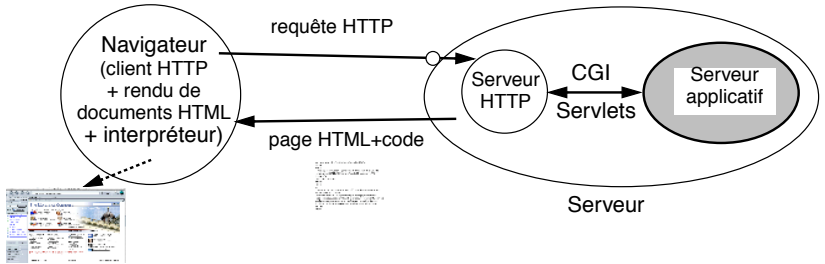
→ Support à la mise en œuvre de systèmes d'informations répartis

→ Architecture à 3 niveaux (3-tier) : client/serveur applicatif/serveur de données

Caractéristiques de WWW

Système ouvert

- basé sur des normes accessibles (disponibles et lisibles par l'homme) :
  URLs (désignation de ressources réparties), HTTP (interaction), HTML (format des contenus)
- diffusion et utilisation large (et/car simple et gratuite) des composants : navigateurs, serveurs
- extensibilité : mécanismes de composition avec du code applicatif



  - ◊ Côté client : ajout de capacités de traitement (interpréteur pour exécuter un code transmis avec le contenu : applets, JavaScript…, plug-ins associés à un contenu (types MIME))
  - ◊ Côté serveur : protocoles de dialogue avec le serveur de l'application : servlets, CGI…
    → pages dynamiques (JSP)
  - ◊ Protocoles spécifiques permis par les URLs
- Généralisation au traitement réparti de données quelconques : XML + Web Services

Caractéristiques de WWW (suite)

Système réparti à large échelle

- Au niveau de l'accès au contenu :

  chaque serveur peut avoir un nombre de clients important et variable

  → problèmes d'efficacité/d'équilibrage de la charge
  - * mise en place de hiérarchies de caches
  - * inadaptation du protocole requête/réponse au suivi de l'évolution des contenus (scrutation)
    → mise en place de schémas publier/s'abonner (RSS)
  - * évolution vers des architectures plus réparties : P2P

- Au niveau de l'utilisateur :
  multiplicité et volatilité des serveurs, des services, des sources d'information
  → comment ne pas se perdre dans l'hyperespace ?

  ◊ Force brute
  - * robots et moteurs de recherche
  - * approche statistique/automatique : observation et analyse individuelle des comportements des utilisateurs

  ◊ Indexation assistée : web sémantique

4 – Intégration d'applications patrimoniales

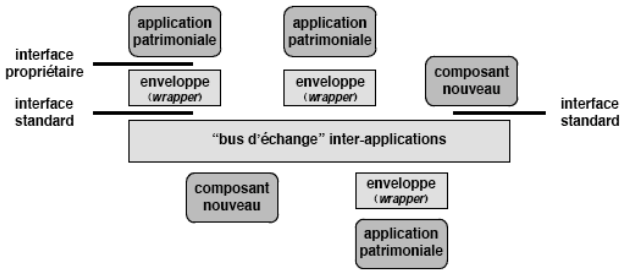
Contexte : développement d'un système d'information global réutilisant un ensemble d'applications existantes et développées indépendamment

- situation « ouverte », comme pour le web : les applications ne sont pas connues a priori, et doivent pouvoir coopéerer sans être retouchées → architecture analogue :

  ◊ utilisation d'une couche d'interaction normalisée, neutre par rapport aux applications (*bus*)

  ◊ définition d'1 norme/modèle communs pour présenter/utiliser les fonctions des applications

- gestion de l'hétérogénéité : ajout d'une couche de conversion/traduction (enveloppe) entre l'interface propriétaire et l'interface commune



Exemples : CORBA, bus à messaages, plateformes d'intégration d'entreprise

5 – Exemple : Télévision interactive

- Fonctions : fourniture d'un ensemble de services aux clients

  ◊ Diffusion de programmes à la demande (individuelle)

  ◊ Télé-achat

  ◊ Jeux interactifs

- Contraintes

  ◊ Interface client familière (télécommande)

  ◊ Disponibilité

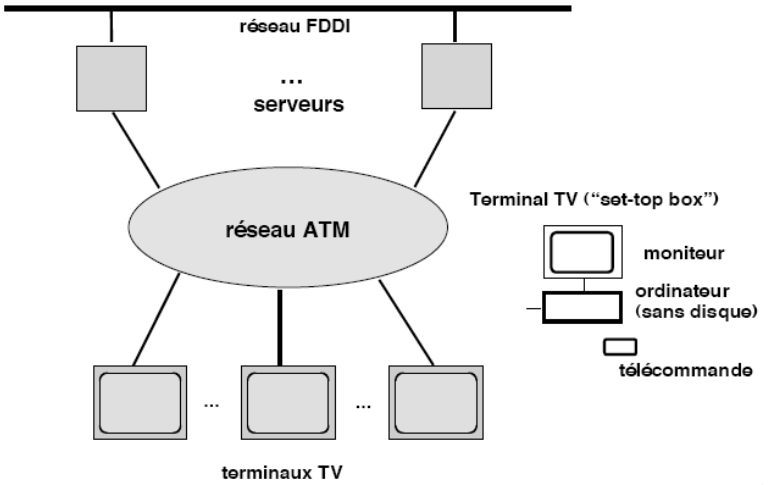
  ◊ Performances

  ◊ Coût (terminal client simple)

  ◊ Extensibilité

Source : M. N. Nelson, M. Linton, S. Owicki, "A Highly Available, Scalable ITV System", Proc. 15th ACM Symp. on Operating Systems Principles, December 1995 [expérience Time-Warner, Orlando]

Architecture



Services et objets

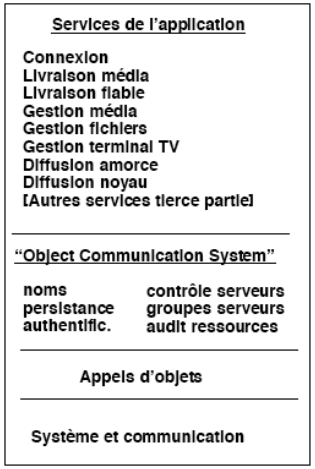
Réalisation des services

1 service = 1 objet

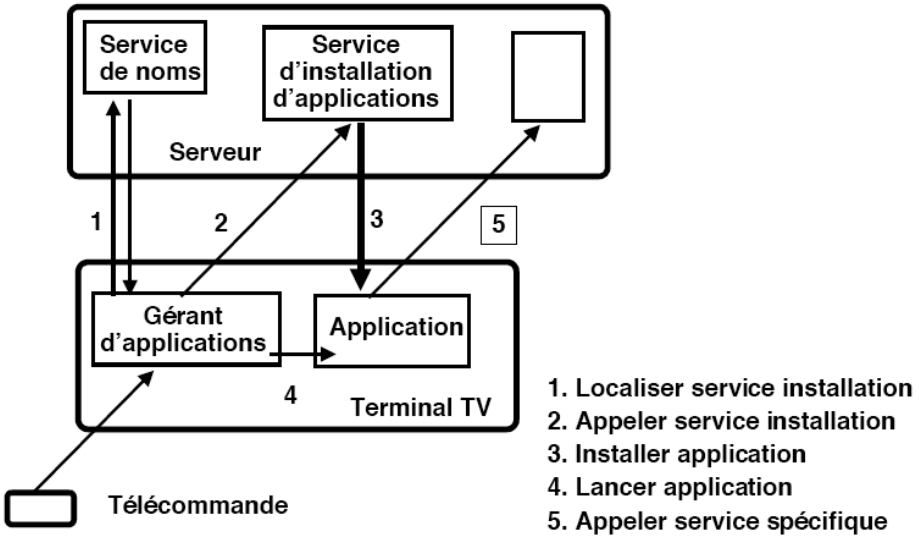
- désignation par référence
- enregistrement dans un service de noms

Disponibilité

- Duplication active
- Serveur primaire/secondaire



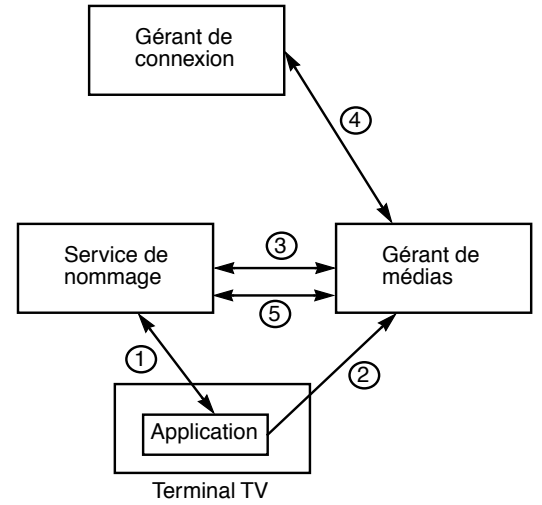
Lancement d'une application



1. Localiser service installation
2. Appeler service installation
3. Installer application
4. Lancer application
5. Appeler service spécifique

Gestion des ressources

Localisation du service



1 - localiser le gérant de médias

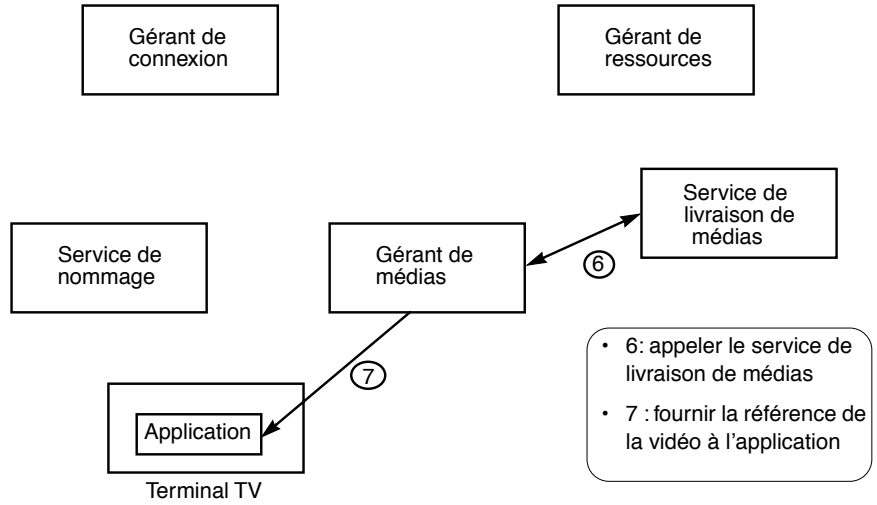
2 - appeler le gérant de médias

3 - localiser le gérant de connexion

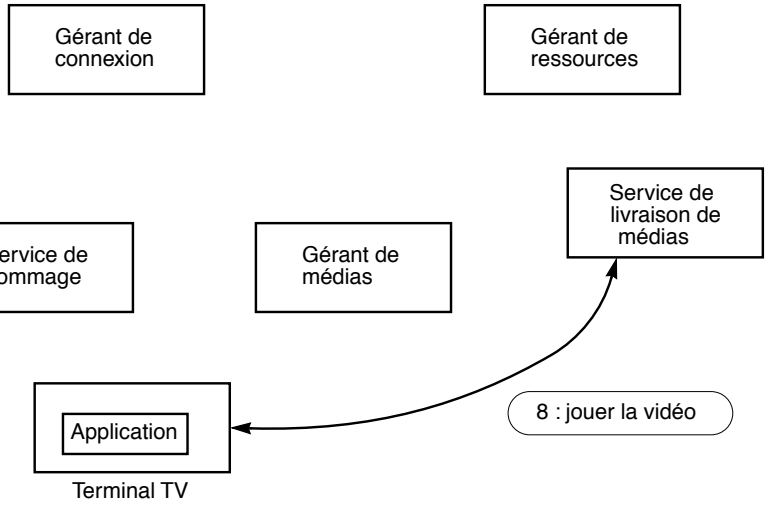
4 - appeler le gérant de connexion

5 - localiser le service de livraison de médias

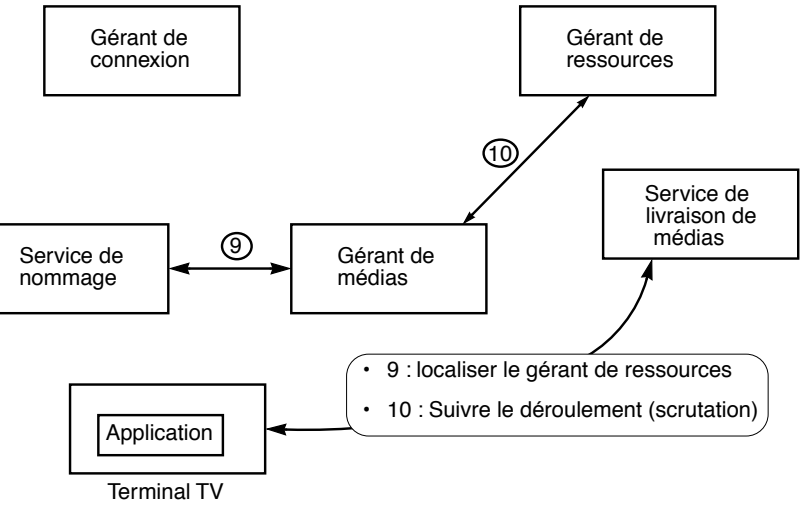
Utilisation du service



- 6: appeler le service de livraison de médias
- 7 : fournir la référence de la vidéo à l'application



Supervision du service



Bilan de l'étude de cas

Contraintes d'une application

- grand public
  - ◊ simplicité de l'interface (terminal TV)
  - ◊ disponibilié
  - ◊ facilité de croissance
- multimédia
  - ◊ respect d'une synchronisation relative au flux
    ex : 16 images/s de 512*512/16 bits = 100 Mb/s → (compression) : 1Mb/s
    → bon dimensionnement et régulation de la charge

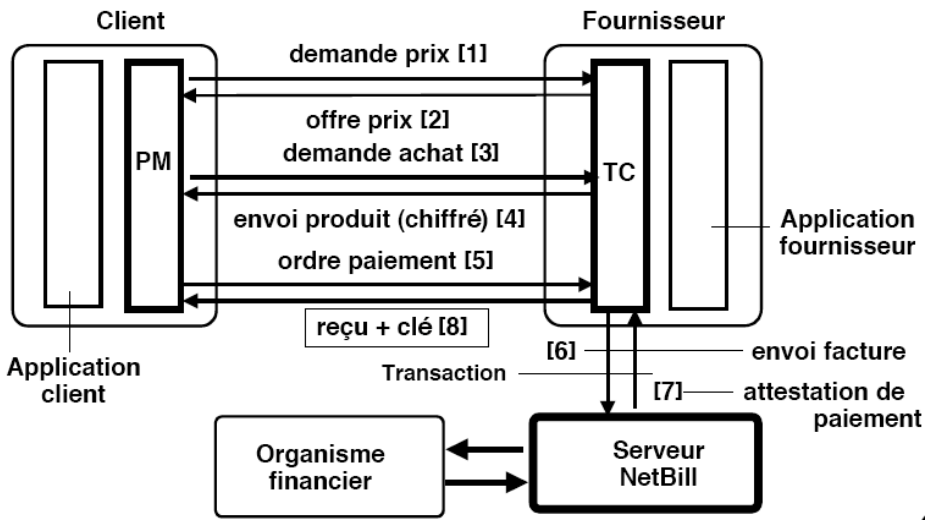
Aspects conception logicielle

- développement modulaire, par assemblage de composants/services logiciels indépendants
- interaction client/serveur
- spécification normalisée des services (IDL) (conception par objets/architecture à composants)
- réalisation de services systèmes communs, sur lesquels s'appuient les services applicatifs :
  - ◊ service de désignation évolué
  - ◊ disponibilité → duplication, reprise

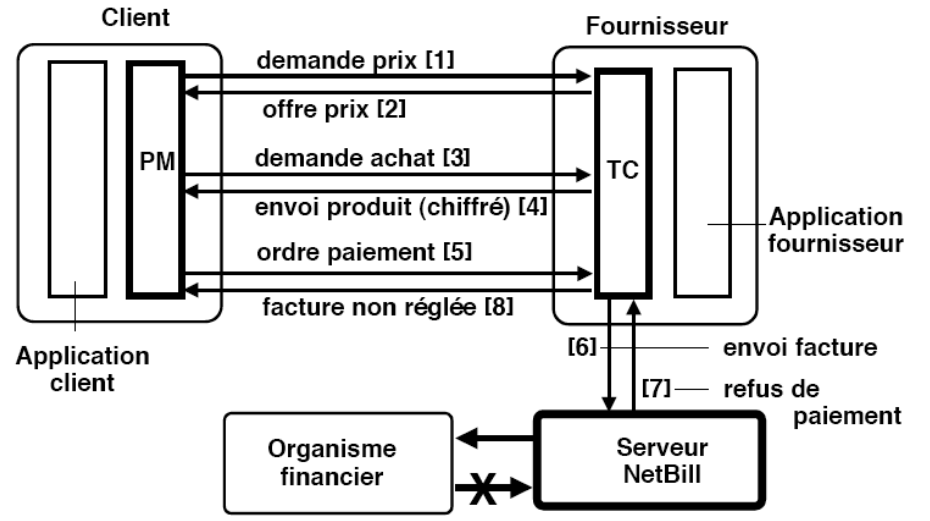
6 – Exemple : Commerce électronique

- Fonction : exécution de transactions commerciales entre clients et fournisseurs
  - ◊ Recherche de produits et services (catalogues)
  - ◊ Commande simple ou groupée
  - ◊ Paiement
  - ◊ Livraison
- Contraintes
  - ◊ Sécurité
    - Protection des informations confidentielles (client et fournisseur)
    - Tolérance aux fautes
    - Intégrité
    - Authentification
  - ◊ Respect des règles de concurrence
  - ◊ Respect des garanties fournisseur->client (offre sincère, exécution du contrat)
  - ◊ Respect des garanties client->fournisseur (identité, paiement)
  - ◊ Respect des droits de propriété (licences, droits d'auteur)
  - ◊ Disponibilité permanente du service

Service de commerce électronique : transaction complète



Service de commerce électronique : transaction incomplète



Tolérance aux fautes

- **Garantie de base** : atomicité de l'évolution du couple (clé disponible, client débité)
- Panne dans les étapes 1 à 4
  - ◊ aucune transaction financière n'a eu lieu
  - ◊ pas de paiement, pas de livraison

Point de vue du client

- Panne après émission de [5] (ordre de paiement)
  - ◊ le client a accepté le produit, mais n'a pas de réponse du serveur
  - ◊ le client a l'initiative de la reprise (contacte fournisseur ou serveur NetBill pour connaître l'état de la requête)
- Erreurs possibles
  - ◊ Ordre de paiement non transmis au serveur : annuler (délai de garde)
  - ◊ Ordre de paiement transmis et accepté (transaction) : le client finira par recevoir la clé (si besoin, du serveur)

Point de vue du fournisseur

- Panne après émission de [6] (facture)
  - ◊ le fournisseur finira par obtenir une réponse du serveur NetBill (au besoin en renvoyant la facture [6])
  - ◊ propriété transactionnelle sur [6]+[7] (envoi facture et résultat du réglement) : annulation possible si panne durable

Bilan de l'étude de cas

Sécurité

- Confidentialité (secret des informations) : chiffrement
- Intégrité (pas de modifications non désirées)
- Authentification
  - ◊ des partenaires (signature électronique)
  - ◊ du contenu des messages

Tolérance aux fautes

- Atomicité des transactions commerciales (paiement+livraison)
- Garanties assurées par le serveur (état défini, opérations transactionnelles)
- Pas d'hypothèses sur sites extérieurs au serveur

7 – Exemple : accès à un fichier distant

Problème

un utilisateur souhaite travailler sur un fichier situé sur une machine distante

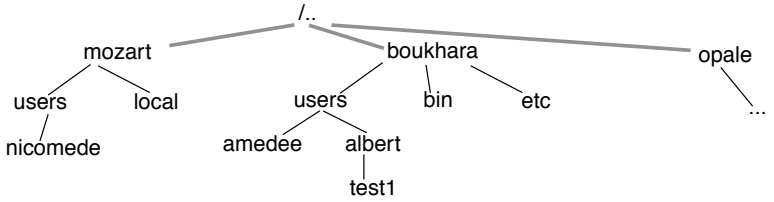
Solutions (par ordre chronologique)

a) Transférer le fichier

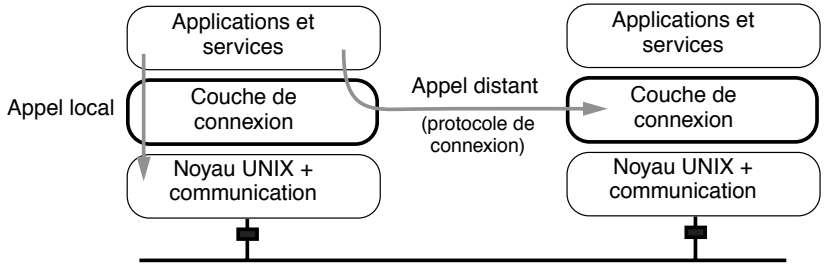
- Faire une copie locale (site utilisateur/client) du fichier distant (site serveur)
- Exemple : via le protocole ftp
  - ◊ envoyer une demande d'ouverture de session (client → serveur)
  - ◊ un processus veilleur (sur le serveur) attend les demandes et établit la liaison avec les clients
  - ◊ le serveur interprète les requêtes d'accès au fichier émises par le client
    si un fichier doit être transmis, un fichier vide est créé sur le site récepteur, puis le fichier est lu et recopié par tronçons de taille fixe → possibilité de contrôles par tronçons
  - ◊ le client termine la session en envoyant une demande de déconnexion
    la liaison virtuelle est alors supprimée
- Solution lourde
  - ◊ volumes échangés
  - ◊ nécessite la connaissance du site et du chemin d'accès au fichier
  - ◊ risques d'incohérence en cas d'accès concurrents

b) Accès explicite à un répertoire distant (Unix United, 1982)

- Extension du mécanisme de désignation de fichiers UNIX par ajout d'une « super-racine » fictive (notée /..) coiffant l'ensemble des sites

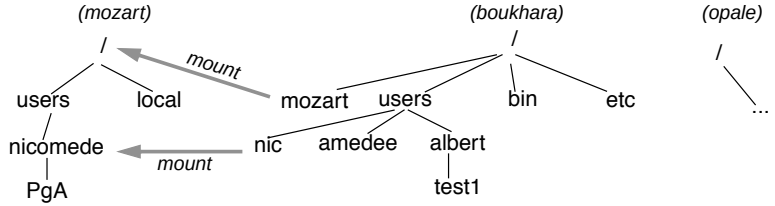


- Mise en œuvre : ajout d'une couche, s'interposant entre le noyau et les applications

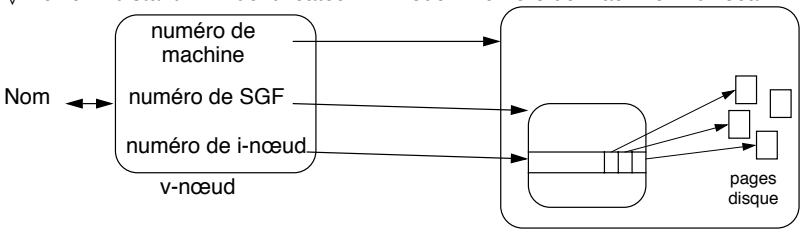


- Accès à l'original du fichier, mais la gestion et la désignation restent lourdes

c) Montage de répertoires (NFS, 1986)



- Exemple : serveur « d'arborescences » (machines sans disque + serveur)
- Mise en œuvre : extension du mécanisme UNIX (ajout d'une indirection)
  - ◊ fichier « local » → identificateur = id. arborescence (SGF) + numéro de i-nœud
  - ◊ fichier « distant » → identificateur = v-node = numéro de machine + id. local



NFS : mise en œuvre (protocole SUN NFS)

- interface du service : RPC
  - ◊ gestion de la correspondance chemin ↔ vnode : démon mountd
    - montage/démontage de répertoires (↔ création/destruction de v-nodes)
    - interprétation des chemins (requête : lookup())
      remarque : interprétation pas à pas (montages en cascade)
  - ◊ opérations sur les fichiers (lire, écrire…) : démons nfsd
  - ◊ destion des caches côté client : démon biod
- choix de mise en œuvre : serveur sans état
  le contexte (position courante…) est fourni à chaque accès
  - ◊ pas d'ouverture/fermeture de v-node
  - ◊ plus robuste (reprise simplifiée en cas de panne { client / serveur)
  - ◊ plus verbeux
  - ◊ côté client : RPCs { temporisés / réitérés
    - possible du fait que les opérations sont idempotentes
    - impossible de distinguer un serveur lent d'un serveur en panne

- réduire les { débits / délais } → utilisation de caches { pages de fichiers / attributs de fichiers }

heuristique : principe de localité (le passé récent est une bonne image du futur proche)

→ conserver localement les données récemment accédées

◊ Problème : cohérence entre copie locale et original

◊ Solutions

- En lecture : durée de validité limitée pour les données des caches

  (fichiers : 3s, répertoires : 30s)

- En écriture : problème, en cas de panne, car serveur sans état
  * écriture immédiate : réduit l'intérêt du cache
  * mémoire secourue
- Solution sûre, mais radicale :
  utilisation d'un service de verrouillage/synchronisation extérieur à NFS

- l'administration reste lourde (statique)

8 – Conclusion

La répartition amène des problématiques nouvelles et difficiles, voire même insolubles

→ prise en charge *automatique* des problèmes spécifiques à la répartition, lorsque c'est possible

→ forme fréquente : *virtualisation*

On présente à l'utilisateur/au programmeur un environnement centralisé virtuel, disponible via une interface locale, implanté dans un environnement réparti.

La répartition est gérée au niveau de l'implantation du service → transparence (norme ANSA)

◊ Interface :
- accès (homogénéité d'interface local/distant),
- localisation (désignation)

◊ Exécution :
- partage (gestion des accès concurrents),
- duplication,
- fautes (masquage des pannes),
- topologie : mobilité, distance (délais), frontières entre opérateurs (législation, facturation)
- échelle (équilibrage de charge, taille des espaces d'adressage, nombre d'objets gérés...)

Remarque : la transparence complète n'est pas forcément toujours possible ou souhaitable (coût)

Systèmes ouverts, à large échelle

- Abstraction et structuration des fonctionnalités d'une application en termes de *services autonomes*, définis par une *interface*

  → Possibilité de composition et d'adaptation dynamiques des services

- Capacité d'évolution : les performances ne doivent pas fluctuer brutalement en cas d'augmentation du nombre d'utilisateurs, de ressources, de sites, ou d'échanges de données.

  → simplifie le développement d'applications, en abstrayant la disponibilité des ressources

  Exemple : DNS

  Contre-exemple : système de numérotation téléphonique

Comment assurer la capacité d'évolution ?

◊ duplication { données (fichiers, caches…) / serveurs }

◊ autonomie des sites

◊ privilégier les décisions locales

→ éviter la prise de décisions basées sur des informations distantes/globales

Systèmes immergés dans le quotidien, souvent sensibles

Sécurité

- Confidentialité
- Intégrité
  ◊ Contrôle des droits d'accès
  ◊ Isolation (pare-feux)
- Authentification, signature électronique
  ◊ Identification des partenaires
  ◊ Non-déni d'envoi ou de réception
  ◊ Messages authentifiés
  ◊ Respect possible de l'anonymat
- Une méthode de base : la cryptographie

Tolérance aux pannes

- Séparation entre machines « contrôlables » (serveurs) et non contrôlables (la plupart)
  → la tolérance aux pannes doit être garantie quoi que fassent les machines non contrôlables
- Technique de base n°1 : la duplication (serveurs, données)
  → protocoles de groupe (appartenance, diffusion)
- Technique de base n°2 : les transactions (atomicité des traitements)

Remarque : la tolérance aux pannes est nécessaire, mais coûteuse
(duplication → communication et synchronisation, transactions → journalisation)

Architecture Client/serveur (C/S)

Protocole et organisation de base depuis l'origine des systèmes répartis

- serveurs départementaux en réseau local → fermes de serveurs de l'informatique en nuage
- modèle de base du Web et de ses applications
  - ◊ pour le public : modèle d'utilisation qui s'est imposé, et devenu naturel
  - ◊ pour le développement d'applications métier : convergence et recouvrement avec
    - les principes de conception logicielles, basés sur la modularité (objets, composants...)
    - les heuristiques de conception de systèmes répartis, basées sur la recherche de décisions aussi locales que possible.
  - ◊ la technologie matérielle et logicielle a suivi (jusqu'à présent) l'évolution des besoins : gestion de caches, protocoles de routage sémantique, multiprocesseurs, multicœurs...

Des architectures alternatives existent

- inconvénients du C/S liés à la centralisation/concentration
  - ◊ fragilité/criticité
  - ◊ coût et limites physiques d'exploitation, qui peuvent devenir bloquants à terme
- alternatives : architectures largement réparties (pair à pair)
  - ◊ pour l'échange de d'informations, mais pas seulement (grilles de calcul...)
  - ◊ robustesse et souplesse
  - ◊ intérêt économique (meilleure utilisation des ressources existantes)
  - ◊ freins : algorithmique plus complexe, problèmes ouverts, résultats d'impossibilité

Plan du cours

- Modèles d'interaction
  - ◊ Interaction par mémoire (virtuelle) partagée
  - ◊ Interaction par messages
    - asynchrone : MOM, JMS
    - synchrone : C/S
  - ◊ Exemple : réalisation de l'interaction C/S avec l'API sockets de Java
- Intergiciels
  - ◊ RPC
  - ◊ RMI
  - ◊ Exemple : réalisation d'une application de messagerie interactive (RMI, JMS)
  - ◊ (Composants)
- Services pour la construction d'applications réparties
  - ◊ Tolérance aux fautes
  - ◊ Données dupliquées
  - ◊ (Transactions
  - ◊ (Sécurité)
  - ◊ (Nommage)

Références

- JM. Geib, Ph Gransart, Ph. Merle : CORBA, des concepts à la pratique. *Interéditions*
  - ◊ clair, synthétique, étude de cas
- R Balter et al : Construction des systèmes répartis. *INRIA, collection didactique*
  - ◊ ancien, mais présentation rédigée, claire et en français des éléments de base.
- S Krakowiak et coll. : Intergiciel et Construction d'Applications Réparties ouvrage basé sur les cours de l'école d'été INRIA "Construction d'Applications Réparties", en ligne, licence libre
  - ◊ Etat de l'art en recherche et développement, centré sur les intergiciels
  - ◊ Présentation synthétique des principales plateformes intergicielles : EJB, .NET, Services Web...
- Cours Mastère de Sacha Krakowiak. *En ligne*
  - ◊ clair, synthétique, complet, (mais conceptuel)
  - ◊ couvre la théorie et l'ingénierie
- J Bacon : Concurrent Systems. *Addison-Wesley*
  - ◊ Aspects système et multiprocesseur
- G Coulouris et al. : Distributed Systems. *Addison-Wesley*
  - ◊ Etat de l'art complet, synthétique et actualisé