Introduction aux systèmes répartis

Frank Singhoff

Bureau C-203

Université de Brest, France

LISyC/EA 3883

singhoff@univ-brest.fr

Sommaire

1. Principes généraux sur les systèmes répartis.
2. Paradigmes et services de communication.
3. Principaux problèmes de coordination et de cohérence de donnée.
4. Conclusion, synthèse.
5. Références.
6. Acronymes.

Sommaire

- Présentation, définitions.
- Principaux problèmes soulevés et services offerts par un système réparti.

Présentation, définitions

- "*Un système réparti est un ensemble de machines autonomes connectées par un réseau, et équipées d'un logiciel dédié à la coordination des activités du système ainsi qu'au partage de ses ressources." Coulouris et al. [COU 94].*

- "*Un système réparti est un système qui s'exécute sur un ensemble de machines sans mémoire partagée, mais que pourtant l'utilisateur voit comme une seule et unique machine." Tanenbaum [TAN 94].*

- Systèmes fortement et faiblement couplés.

- Notion d'**image unique du système** $\Longrightarrow$ système généralement incomplet.

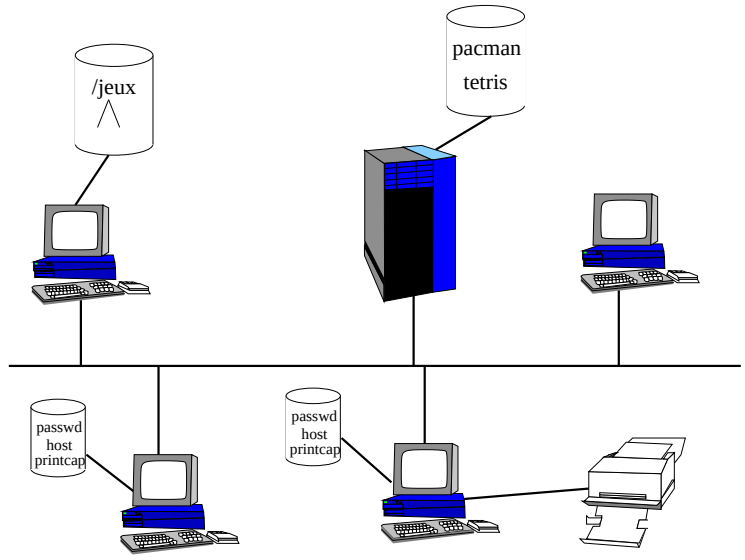
Pourquoi un système réparti ?

1. Partage des ressources (données, applications, périphériques chers). Optimisation de leur utilisation (ex : au LIP6, machines et processeurs libres 69% et 93% du temps).
2. Tolérance aux pannes (fiabilité, disponibilité).
3. Contraintes physiques (ex : avionique, usines automatisées).
4. Interconnexion de machines dédiées (ex : MVS-CICS + PC windows).
5. Facilite la communication entre utilisateurs.
6. Concurrence, parallélisme $\Longrightarrow$ efficacité.
7. Prix des processeurs de petite puissance inférieur à ceux de grande puissance $\Longrightarrow$ raisons économiques.
8. Flexibilité, facilité d'extension du système (matériels, logiciels). Sauvegarde de l'existant.

Ce qu'offre un système réparti

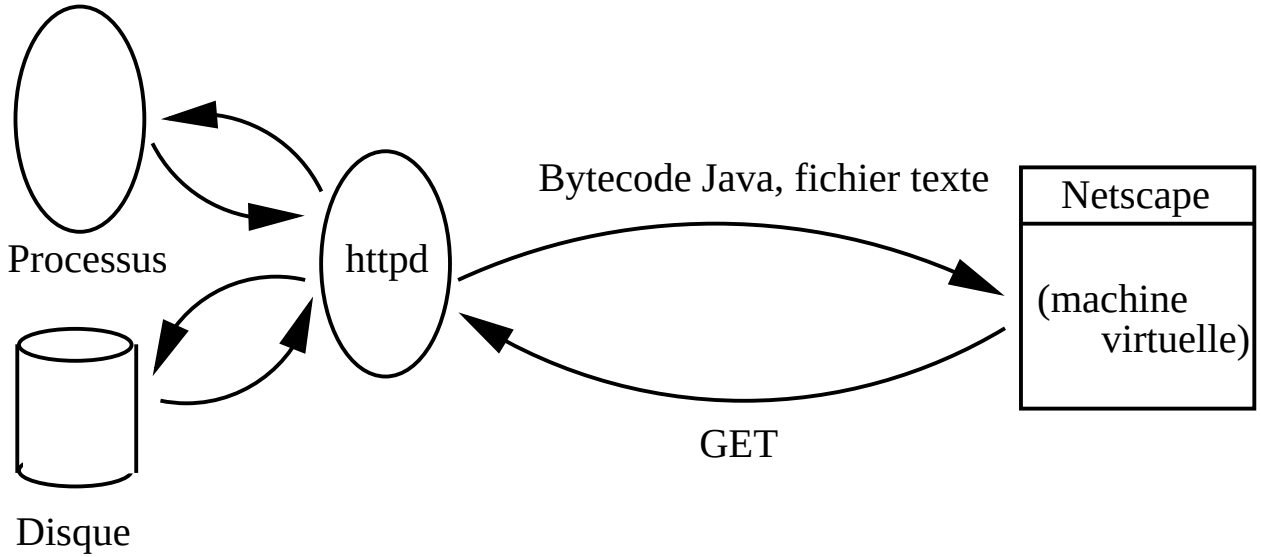
1. **Transparence à la localisation** $\Longrightarrow$ désignation. L'utilisateur ignore la situation géographique des ressources. Transparence à la migration.
2. **Transparence d'accès**. L'utilisateur accède à une ressource locale ou distante d'une façon identique.
3. **Transparence à l'hétérogénéité**. L'utilisateur n'a pas à se soucier des différences matérielles ou logicielles des ressources qu'il utilise. **Notion d'interopérabilité**.
4. **Transparence aux pannes** (réseaux, machines, logiciels). Les pannes et réincarnations sont cachées à l'utilisateur. Transparence à la réplication.
5. **Transparence à l'extension des ressources**. Extension ou réduction du système sans occasionner de gène pour l'utilisateur (sauf performance).

Exemple 1 : le système UNIX



- Transparence d'accès et à la localisation : NFS, le service d'impression. NIS : transparence à la localisation mais pas d'accès ($ypcat$).
- Désignation : nom de machine + port. Pas indépendante de l'adressage.
- Interopérabilité : RPC (XDR), IP.

Exemple 2 : le Web



- Transparence à la localisation : liens hypertexte.
- Transparence d'accès : URL (site $\neq$ adresse IP).
- Désignation : URL, DNS Internet.
- Interopérabilité : Java bytecode et machine virtuelle, pages HTML, Web Services (J2EE, .NET).

Sommaire

- Présentation, définitions.
- Principaux problèmes soulevés et services offerts par un système réparti.

Cohérence et synchronisation (1)

- **Architecture :**
  1. Centralisée = 1 horloge. Ordre total.
  2. Répartie = plusieurs horloges non synchronisées + communications **asynchrones** $\Longrightarrow$ plus d'**état global** facilement calculable et présence d'**indéterminisme logique**.

- **Problèmes :** partage de données réparties et coordination répartie.

- **Solutions :** algorithmes dédiés d'élection, d'exclusion mutuelle, de consensus, de terminaison, etc.

Cohérence et synchronisation (2)

- **Exemple 1 : partage de données réparties**
  - Echanges d'email $\Longrightarrow$ consensus réparti
  - Réponses reçues avant les questions.

- **Exemple 2 : coordination répartie**
  - Mise au point d'un programme : comment arrêter plusieurs processus répartis simultanément ? $\Longrightarrow$ synchronisation.
  - Comment réexécuter une application ? $\Longrightarrow$ indéterminisme logique.

Désigner les ressources du système

- *"Désigner un objet consiste à lui affecter un nom permettant de lui faire référence" [MAR 88].*

  - Nom indépendant de la localisation géographique (pannes, migrations) et du temps.
  - Nom = ressource $\Longrightarrow$ réutilisation.
  - **Adresse** $\neq$ **nom**. Translation nom $\Longrightarrow$ adresse = service de localisation (chaîne de références, diffusion, fonctionnelle, serveur de localisation).
  - Espace de nommage/adressage, contexte (ex : désignation hiérarchique).
  - Serveur centralisé ou par type de ressource (système de fichiers) ; performance, résistance aux pannes, facilité de réalisation, répartition ou non du catalogue (cohérence).

Support de l'hétérogénéité

- **Services interopérables :** comment s'affranchir des différences matérielles et logicielles des machines du système ?

- **Sources d'hétérogénéité :**

1. La représentation des données en mémoire :
   - Le problème des petits indiens (systèmes big indian et little indian).
   - Représentation des flottants (ex : PC sur 80 bits, norme IEEE en 64 et 128 bits).
   - Entiers complémentés à 1 ou 2.
   - Alignement des données en mémoire.
2. Les langages de programmation, les exécutables, les systèmes d'exploitation.
3. Les protocoles de communication.

Transparence aux pannes (1)

- **Objectif :** cacher à l'utilisateur l'occurrence de pannes.

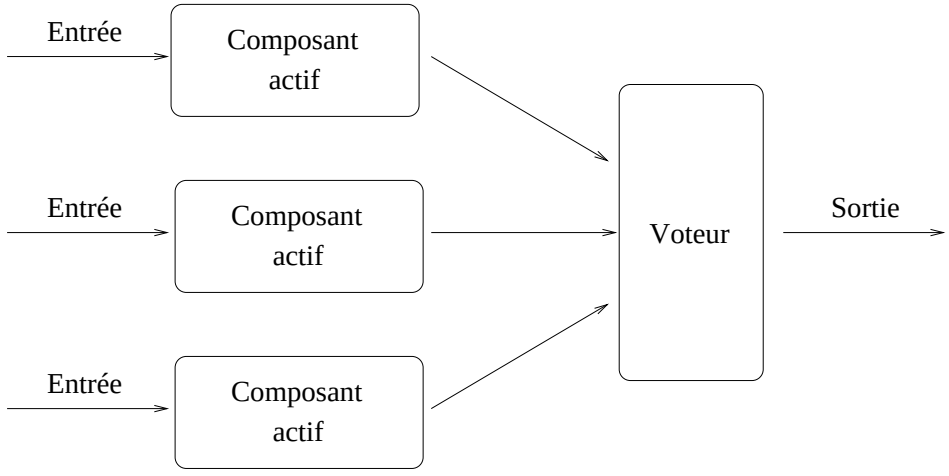
Reste un problème difficile dans un système asynchrone.

- *Un système distribué est un système avec lequel je ne peux rien faire car une machine que je ne connais pas est en panne". Lamport.*

  - Type de pannes : pannes temporelles, pannes dites "silencieuses" (crash), pannes byzantines.
  - Solutions par points de reprise globaux $\Longrightarrow$ comment prendre un état cohérent du système (message du futur).
  - Solutions à base de redondance.

Transparence aux pannes (2)

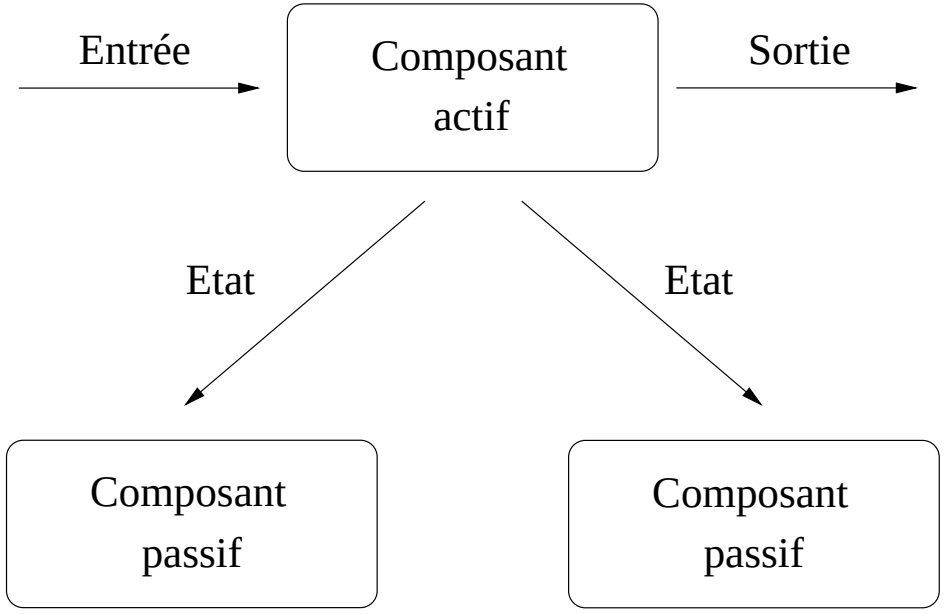
- **Redondance active :**



- Groupe de serveurs + vote (consensus).
- Application déterministe.
- Consensus impossible dans un système asynchrone.

Transparence aux pannes (3)

- **Redondance passive :**



- Maître/esclaves (journalisation).
- Application non déterministe mais systèmes coûteux.
- Quoi et comment journaliser ?

Autres problèmes importants

- Sécurité. Accès aux données confidentielles. Authentification. Utilisation des ressources (périphériques, logiciels licenciés). Malveillance.
- Performance (nécessite de faire au moins aussi bien qu'un système centralisé) $\Longrightarrow$ mécanismes ajoutés pour la tolérance aux pannes, la sécurité, la transparence, etc.
- Passage à l'échelle (algorithmes centralisés).
- Administration (ex : NFS).
- Etc.

Résumé : services offerts

- En plus des services classiquement rencontrés dans un système, le système réparti doit donc offrir :

  - Services de désignation et de localisation.
  - Services de communication.
  - Services de synchronisation.
  - Services de tolérance aux pannes.
  - Services de sécurité.

Aspects architecturaux

- Où se trouvent ces services ?
  - Dans la couche transport (UNIX + TCP/IP).
  - Dans le langage de programmation (Ada 95, Java RMI).
  - Dans le "Middleware"/intergiciel (CORBA, RPC, .NET). Logiciel se situant entre le système d'exploitation et les applications.
  - Dans le système d'exploitation (Chorus, Mach, Amoeba) $\Longrightarrow$ micro-noyau.

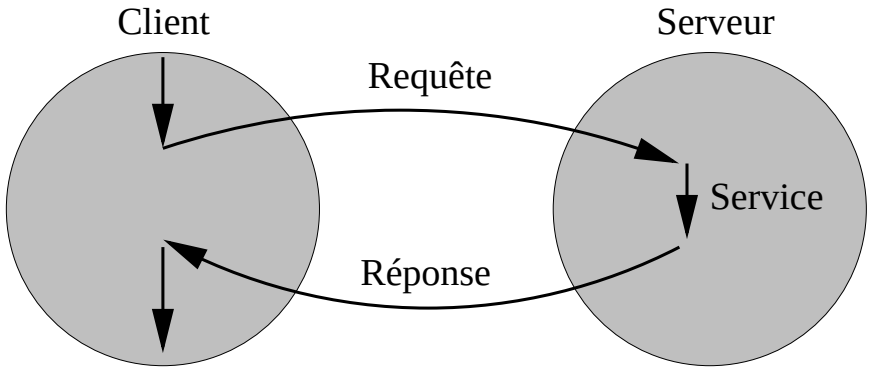
Sommaire

1. Principes généraux sur les systèmes répartis.
2. Paradigmes et services de communication.
3. Principaux problèmes de coordination et de cohérence de donnée.
4. Conclusion, synthèse.
5. Références.
6. Acronymes.

Sommaire

1. Paradigmes de coopération.
2. Services de communication.

Paradigme client/serveur (1)



- Notion de client, service et serveur (ex : service d'impression, de base de données) [ORF 95].
- Interaction synchrone entre le client et le serveur : le client est bloqué tant que le serveur n'a pas répondu.
- Serveur concurrent (processus, thread) ; ex : ftp.
- Serveur avec ou sans état.
- Extensibilité, intégration de produits divers.
- Diffusion, flots de données.

Paradigme client/serveur (2)

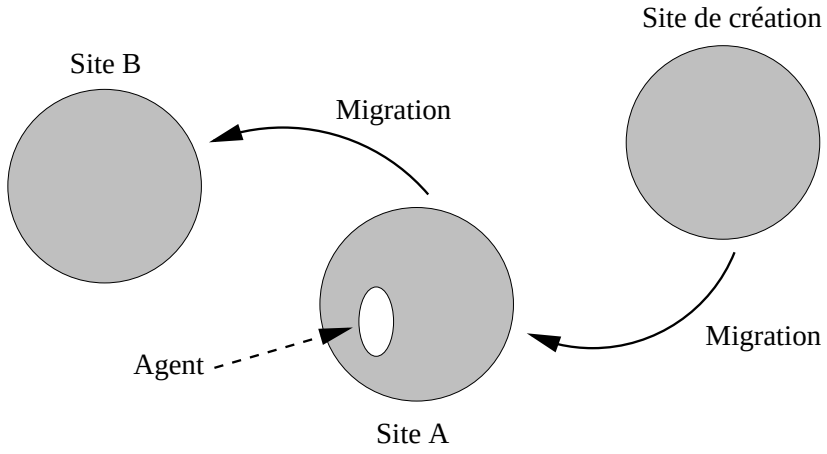
- Client/serveur : extension naturelle de la notion de service dans un environnement centralisé ... mais.

- Pannes possibles : perte de la requête ou de la réponse, panne du serveur ou du client.

- Notion de sémantique des opérations :
  - Exactement une fois.
  - Au moins une fois (opération idempotente).
  - Au plus une fois.

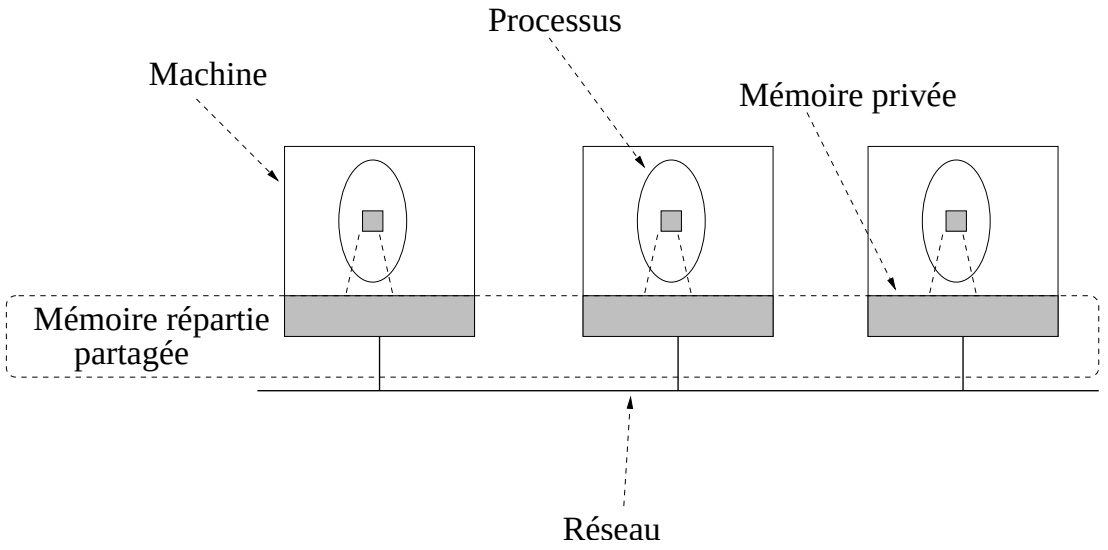
- Réincarnation du serveur si serveur avec état (synchronisation, journalisation).

Paradigme des agents mobiles



- Agent autonome se déplaçant au grès des machines pour réaliser ses tâches [BER 99]. Possède un état.
- Asynchrone $\Longrightarrow$ mieux adapté que le client/serveur à des traitements longs.
- Exemple : recherche sur le Web, administration réseaux $\Longrightarrow$ moins de transfert de données.
- Nouveaux problèmes : sécurité (de l'agent, de l'hôte), mobilité du code, désignation.

Paradigme de la mémoire partagée



- Transparence d'accès et à la localisation.
- Persistance possible. Ramasse miettes.
- Données partageables : variables, pages, objets. Copie en lecture, migration en écriture. Problème de cohérence.
- Adapté pour les applications de calculs parallèles. Efficacité ?

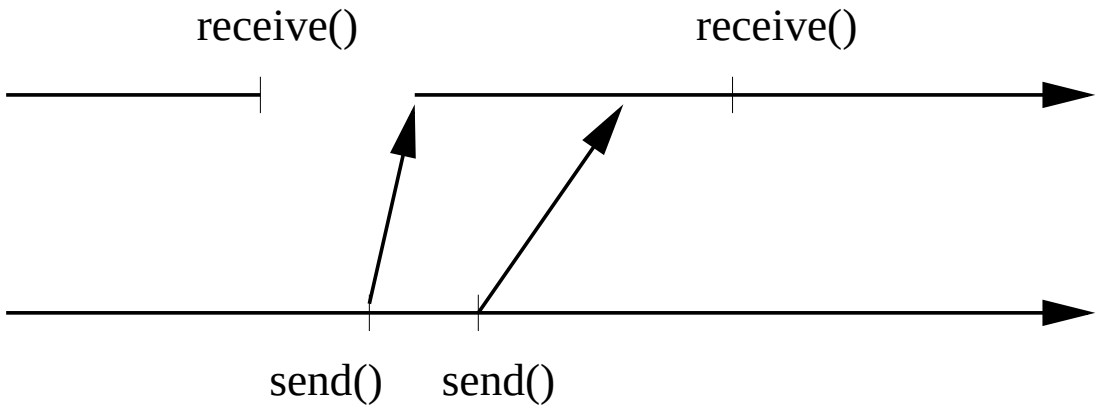
Sommaire

1. Paradigmes de coopération.
2. Services de communication.

Services disponibles

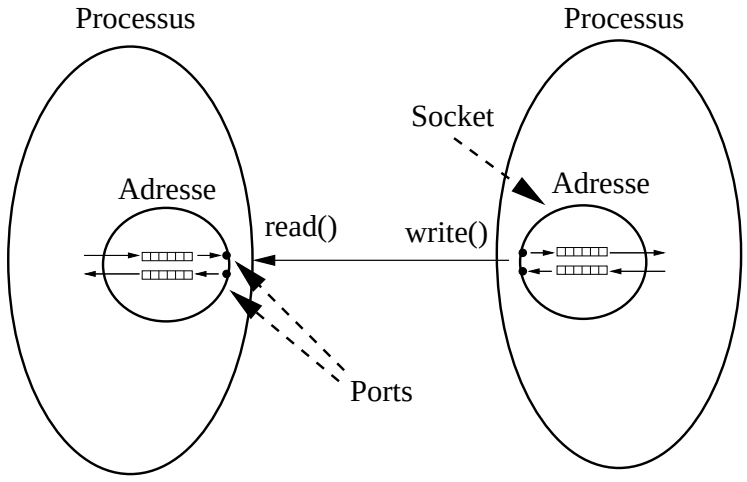
- Echange de messages asynchrones.
- Services synchrones.
- Des services synchrones aux systèmes à objets répartis.

Messages asynchrones



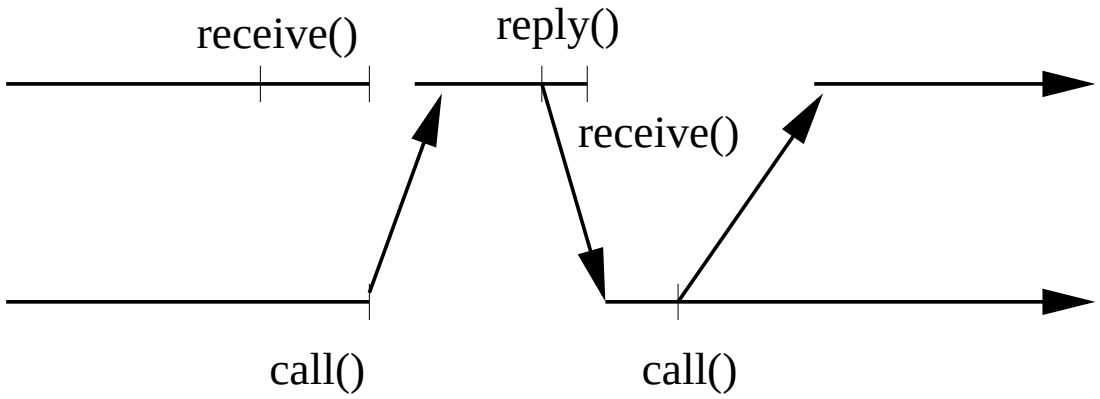
- C’est la brique de base. Mode le plus rencontré.
- Producteur/consommateur. Asynchrone $\Longrightarrow$ tampon.
- Perte, duplication, déséquencement.
- Bloquant ou non bloquant.
- Ordre total local + ordre causal.
- Lourd et compliqué pour le développeur (il doit assurer toute la synchronisation) mais puissant et souple.

Messages asynchrones : UDP



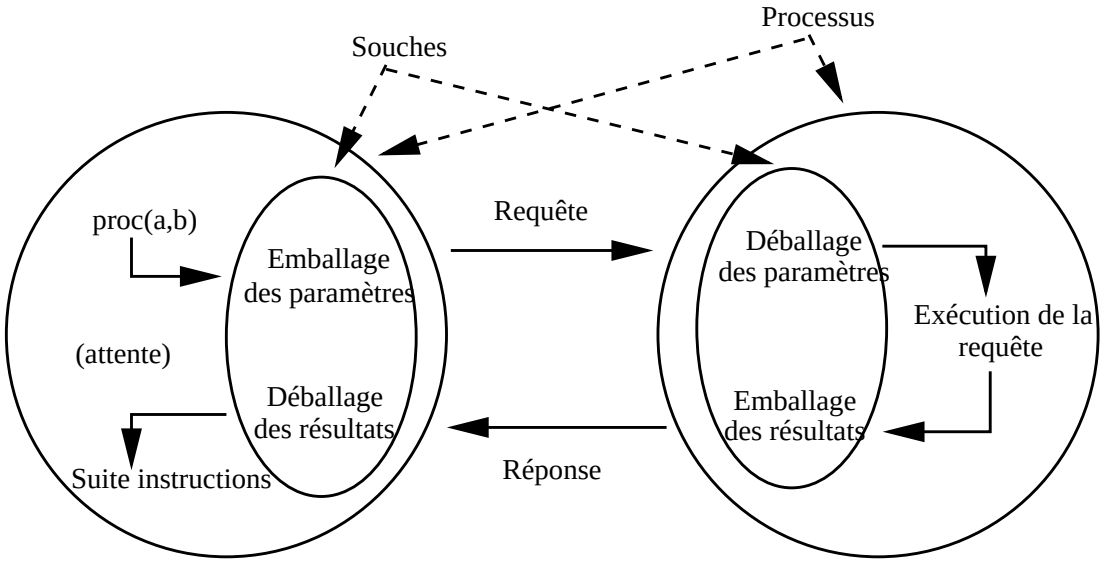
- Sockets UDP (connectées ou non) = tampons.
- Lecture et écriture bloquante ou non.
- UDP = datagramme non fiable.
- Pas de désignation mais adressage. Pas de transparence d'accès et à la localisation. Pas de gestion de l'hétérogénéité.
- Exemple de service : NFS $\Longrightarrow$ d'où serveur sans état.

Communications synchrones



- Contrairement au message asynchrone, le protocole est maintenant asymétrique : notion de client et de serveur.
- Le client est bloqué jusqu'à la réponse du serveur.
- Propriété d'ordre plus forte (ordre total sur les requêtes d'un même client).
- Généralement réalisées par messages asynchrones
- Sémantique d'invocation si basé sur une couche transport non fiable.

Appel de procédures à distance (1)



- Structure le programme de façon familière pour le programmeur.
- Communication transparente à l'utilisateur
- Prise en compte de l'hétérogénéité : sérialisation et encodage des données.
- Notion de souches : encodage/décodage + communication. Souches souvent générées.

Appel de procédures à distance (2)

- Exemple : les RPC SUN [RIF 95].
  - Hétérogénéité grâce aux filtres XDR ; sérialisation de structures complexes.
  - Description des données en XDR, puis, utilisation du compilateur $rpcgen$ (génération des filtres, sources, programme principal).
  - Pas de transparence d'accès (signature des procédures), et pas de transparence à la localisation (numéro de programme, version, procédure + adresse IP). Service $portmap$ pour le port.
  - Exemple d'utilisation : NFS, NIS.
  - Possibilité d'invocation asynchrone (sans résultat).

Le modèle à objets répartis (1)

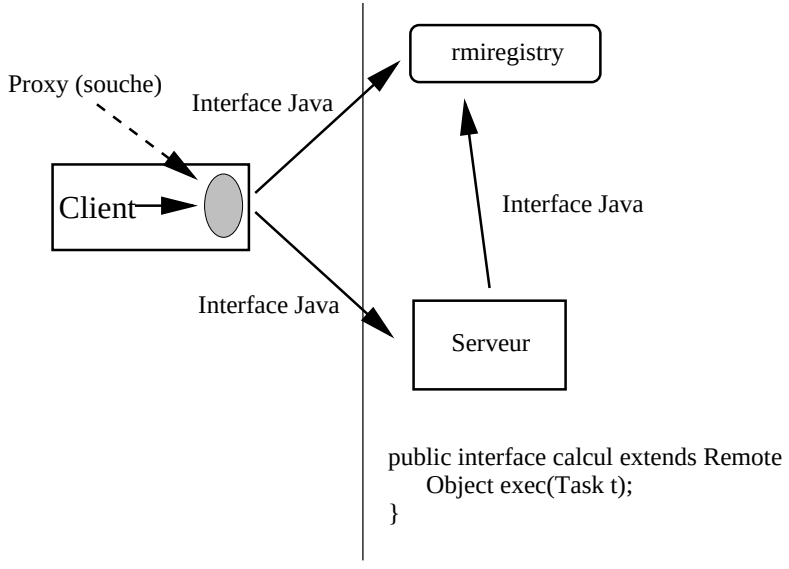
- Extension de l'appel de procédure à distance dans le monde objet.

- Objet = code + données (unité d'encapsulation). Ce sont très souvent des serveurs à état.

- Transparence à l'hétérogénéité, à la localication, à l'accès :
  - Inter-opérabilité (représentation des données, langages de programmation, protocoles et systèmes).
  - Notion de proxy (design-pattern) : représentation locale d'un objet distant. Génération automatique de ces proxies.
  - Génération des composants logiciels par compilation d'interface.
  - Service de désignation/nommage, d'événements (design-pattern).

Le modèle à objets répartis (2)

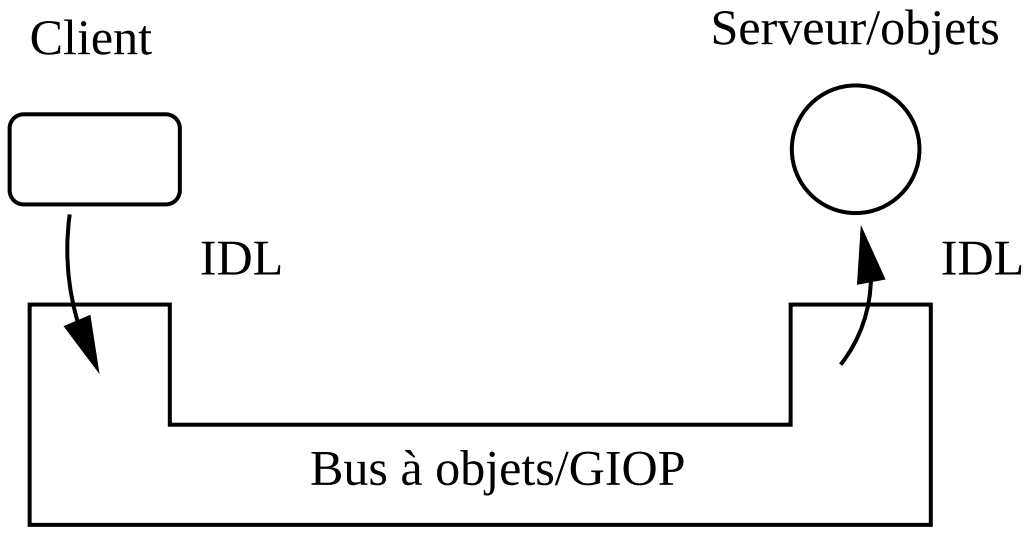
- Exemple 1 : Java RMI



1. Quasi-totale transparence d'accès $\Longrightarrow$ service décrit par une interface Java. Pas de transparence à la localisation.
2. Proxy généré par $rmic$, sérialisation.
3. Service de désignation obligatoire ($rmiregistry$).
4. Protocoles de communication : JRMP ou GIOP (CORBA).
5. Support de l'interopérabilité ... limité à Java sauf GIOP.

Le modèle à objets répartis (3)

- Exemple 2 : CORBA



1. Transparence à la localisation mais pas de transparence d'accès $\Longrightarrow$ service décrit en IDL et API spécifique.
2. Générateur IDL $\Longrightarrow$ utilisation de proxy mais approche plus statique.
3. Service de désignation optionnel.
4. Protocole de communication (bus) : GIOP.
5. Support de l'hétérogénéité plus étendu. Portabilité, intégration de service.

Sommaire

1. Principes généraux sur les systèmes répartis.
2. Paradigmes et services de communication.
3. Principaux problèmes de coordination et de cohérence de donnée.
4. Conclusion, synthèse.
5. Références.
6. Acronymes.

Cohérence et synchronisation (1)

- **Architecture :**
  1. Centralisée = 1 horloge. Ordre total.
  2. Répartie = plusieurs horloges non synchronisées + communications **asynchrones** $\Longrightarrow$ plus d'**état global** facilement calculable et présence d'**indéterminisme logique**.

- **Problèmes :** partage de données réparties et coordination répartie.

- **Solutions :** algorithmes dédiés d'élection, d'exclusion mutuelle, de consensus, de terminaison, etc.

Cohérence et synchronisation (2)

- **Exemple 1 : partage de données réparties**
  - Echanges d'email $\Longrightarrow$ consensus réparti
  - Réponses reçues avant les questions.

- **Exemple 2 : coordination répartie**
  - Mise au point d'un programme : comment arrêter plusieurs processus répartis simultanément ? $\Longrightarrow$ synchronisation.
  - Comment réexécuter une application ? $\Longrightarrow$ indéterminisme logique.

Sommaire

- **Algorithmes de coordination/synchronisation et de partage de données réparties :**

1. Algorithmes centralisés.
2. Algorithmes répartis.
3. Construction d'horloges globales.

Algorithmes centralisés vs répartis

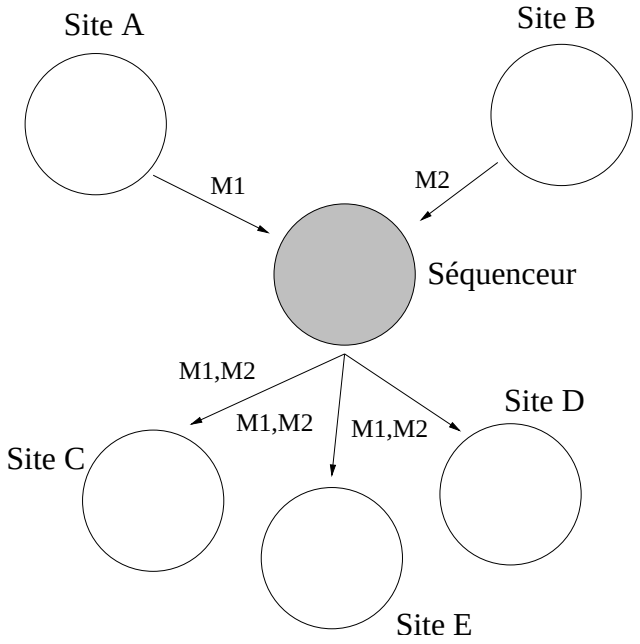
- Facilité de réalisation, comportement déterministe.
- Solutions asymétriques $\Longrightarrow$ déploiement, administration (placement).
- Goulot d'étranglement : performance, tolérance aux pannes.

vs

- Généralement symétrique.
- Flexibilité (administration). Tolérance aux pannes.
- Passage à l'échelle. Performance ? (complexité).
- Très difficiles à mettre au point $\Longrightarrow$ effet de sonde, indéterminisme.

Algorithme centralisé (1)

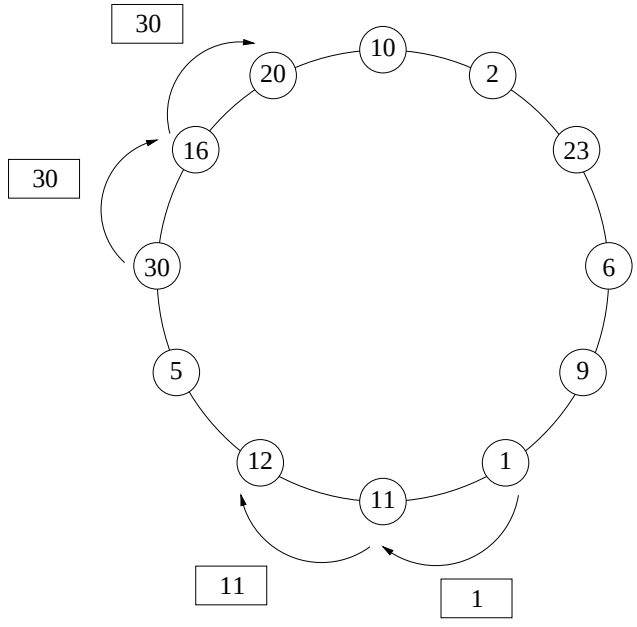
- Exemple : le séquenceur (diffusion atomique avec ordre total).



- Algorithme simple utilisable pour toutes les ressources (exclusion mutuelle, partage de données ou de périphériques).
- Exemple : NFS.

Algorithme centralisé (2)

- Algorithme d'élection de Chang et Roberts [CHA 79] :



- Boucle physique ou logique. Initiateur multiple.
- Réception d'un message : (1) site<message $\Longrightarrow$ transmission ; (2) site>message $\Longrightarrow$ modification du message.
- Rotation de confirmation.

Algorithme réparti (1)

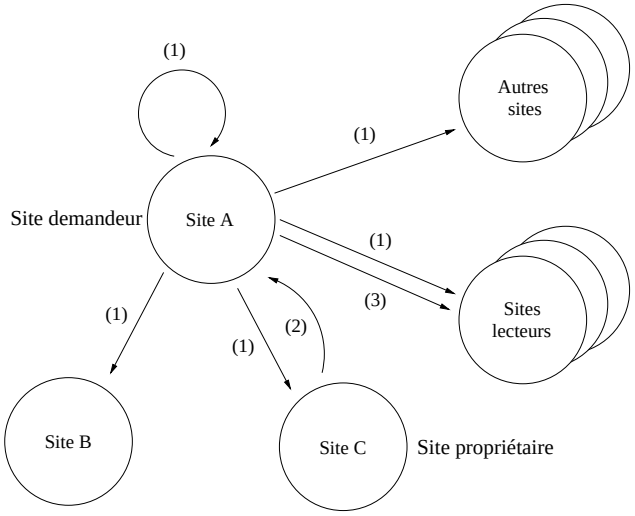
- Algorithme de Li et Hudak [LI 89].
- Plusieurs sites se partagent l'accès à un groupe de pages de mémoire.
- **Applicable sur toute forme de donnée, exclusion mutuelle** : pages, variables, fichiers, ...
- Implante une cohérence forte des données $\Longrightarrow$ lecture = dernière écriture.

Algorithme réparti (2)

- **Chaque site maintient une table de pages, comprenant pour chaque page :**
  - Le propriétaire (Owner) de la page = dernier rédacteur.
  - Le type d'accès (Access) du site sur la page (lecture, écriture ou nil).
  - Liste des lecteurs (Copyset) : liste des sites possédant une copie en lecture $\Longrightarrow$ invalidations lors des écritures
  - Un sémaphore pour protéger les accès à la table de pages.

Algorithme réparti (3)

- **Algorithme par diffusion, requête en écriture :**



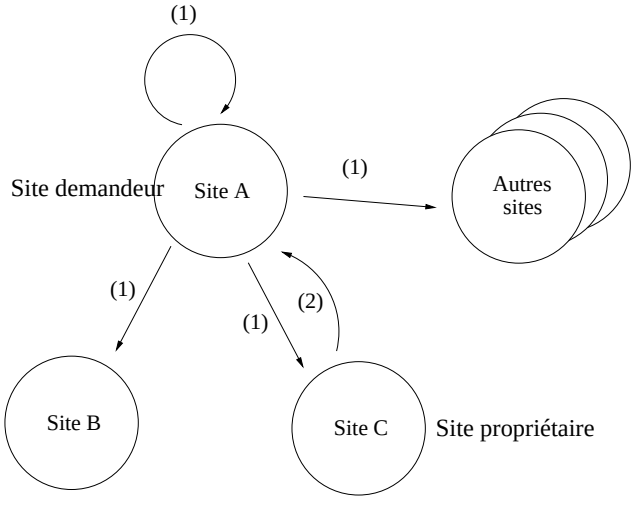
(1) Le site A diffuse sa requête à tous les sites.

(2) Le site C, propriétaire, envoie la page et le copyset au demandeur, A.

(3) Le site A envoie des invalidations aux sites lecteurs (détenteurs d'une copie de la page concernée en lecture).

Algorithme réparti (4)

- **Algorithme par diffusion, requête en lecture :**



(1) Le site A diffuse sa requête à tous les sites.

(2) Le site C, propriétaire, envoie une copie de la page et insère le site A dans le copyset.

Algorithme réparti (5)

Gestionnaire de défauts en lecture

Verrouiller (PTable[p].sémaphore)
diffuser une demande en lecture pour p
attendre la réception de p
PTable[p].accès := lecture
Déverrouiller (PTable[p].sémaphore)

Serveur en lecture

Verrouiller (PTable[p].sémaphore)
Si je suis le propriétaire de p
Alors
    PTable[p].copyset := PTable[p].copyset ⋃ {s}
    PTable[p].accès := lecture
    Envoyer p au site s
FSi
Déverrouiller (PTable[p].sémaphore)

Algorithme réparti (6)

Gestionnaire de défauts en écriture

Verrouiller (PTable[p].sémaphore)
Diffuser une demande en écriture pour p
Attendre la réception de p et de son copyset
Pour tout i dans copyset Faire
   Envoyer à i Invalidation (p)
PTable[p].accès := écriture
PTable[p].copyset := ∅
PTable[p].propriétaire := ego
Déverrouiller (PTable[p].sémaphore)

Algorithme réparti (7)

Serveur en écriture

Verrouiller (PTable[p].sémaphore)
Si je suis le propriétaire de p
Alors
    Envoyer p et PTable[p].copyset au site s
    PTable[p].accès := nil
    PTable[p].propriétaire := s
FSi
Déverrouiller (PTable[p].sémaphore)

Serveur d'invalidation

Verrouiller (PTable[p].sémaphore)
PTable[p].accès := nil
Déverrouiller (PTable[p].sémaphore)

Ordonner les événements (1)

- Systèmes répartis : pas d'horloge globale.

- Production d'une horloge dans le système réparti = imposer un ordre des événements.

  $\Longrightarrow$ Utilisation d'horloges logiques [RAY 96]. Basées sur les communications.

Ordonner les événements (2)

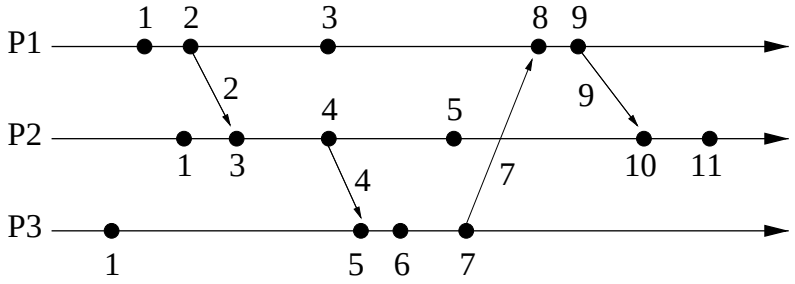
- Notion d'événements pertinents pour une application donnée.

- Observation de la progression des autres processus du système grâce à la notion de **causalité** [LAM 78]. Ordre partiel.

- x **est causalement dépendant de** y **(noté** $x \Rightarrow y$**) si :**
  1. x et y sont des événements locaux (dits internes) et x s'exécute avant y.
  2. x et l'émission d'un message et y sa réception sur un site différent.
  3. Si $x \Rightarrow z$ et $z \Rightarrow y$ alors $x \Rightarrow y$.

- $x \Rightarrow y$ signifit que x précède temporellement y.

Ordonner les événements (3)



- Un compteur par site (H_i).
- Mise à jour de l'horloge locale :
  1. Réception message : $H_i = max(H_i, H_{msg})$.
  2. Autres événements : $H_i = H_i + 1$.
- Horloge globale : ordre total de Lamport (noté $\prec$) ; soient deux événements, x sur le site i, y sur j :

$$x \prec y \Leftrightarrow [H_x < H_y \quad ou \quad (H_x = H_y \; et \; i < j)]$$

- **L'ordre total de Lamport masque la causalité (ex : evt 3).**

Ordonner les événements (4)

- **Comment connaître la progression des autres sites :**
  $\Longrightarrow$ horloges vectorielles [FID 91, MAT 89, SCH 88]
  - Un vecteur de compteurs par site ($V[1...n]$).
    $V[i]$ = horloge du site i. V = vue des horloges des autres sites.
  - Mise à jour de l'horloge locale :
    1. Réception message :

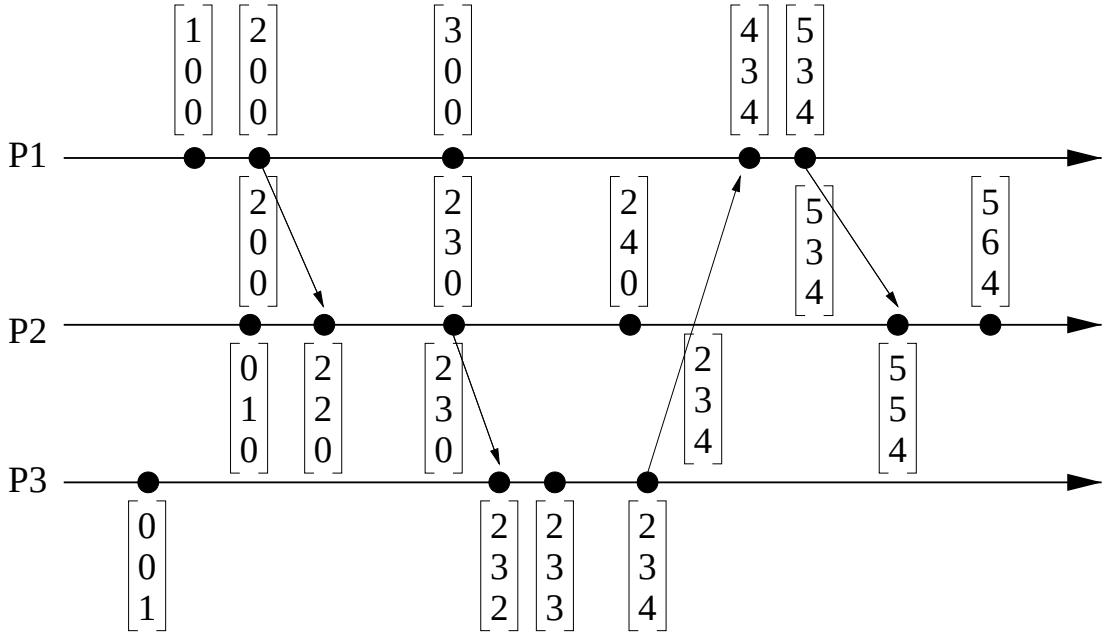
$$\forall\ k\ avec\ k \neq i : V[k] = max(V[k], msg[k])$$

    2. Pour tous les autres événements : $V[i] = V[i] + 1$.

- Horloges coûteuses mais capture l'ordre causal.

Ordonner les événements (5)

- Exemple :



Ordonner les événements (6)

- Ordre causal événements x et y tel que VX et VY soient leur vecteur d'estampilles :

$$\begin{aligned} x \Rightarrow y &\Leftrightarrow VX < VY \\ x \parallel y &\Leftrightarrow VX \parallel VY \end{aligned}$$

Avec :

$$\begin{aligned} VX \leq VY &\Leftrightarrow \forall\, i \;:\; VX[i] \leq VY[i] \\ VX < VY &\Leftrightarrow (VX \leq VY) \;\wedge\; (\exists\, i \;:\; VX[i] < VY[i]) \\ VX \parallel VY &\Leftrightarrow \neg(VX < VY) \;\wedge\; \neg(VY < VX) \end{aligned}$$

Sommaire

1. Principes généraux sur les systèmes répartis.
2. Paradigmes et services de communication.
3. Principaux problèmes de coordination et de cohérence de donnée.
4. Conclusion, synthèse.
5. Références.
6. Acronymes.

Conclusion, synthèse et perspectives

- "*Un système distribué est un système qui s'exécute sur un ensemble de machines sans mémoire partagée, mais que pourtant l'utilisateur voit comme une seule et unique machine.*" Tanenbaum [TAN 94].

  - Objectifs : partage de ressources , tolérance aux pannes, coûts, contraintes physiques.
  - Services : communication, désignation, hétérogénéité, synchronisation et coordination.

- Un mot clef important à retenir : ***Transparence***

$\Longrightarrow$ partielle actuellement.

Sommaire

1. Principes généraux sur les systèmes répartis.
2. Paradigmes et services de communication.
3. Principaux problèmes de coordination et de cohérence de donnée.
4. Conclusion, synthèse.
5. Références.
6. Acronymes.

Références

[BER 99] G. Bernard. « Applicabilité et performances des systèmes d'agents mobiles dans les systèmes répartis ». pages 57–68. 1ère Conférence française sur les systèmes d'exploitation, juin 1999.

[CHA 79] E. G. Chang and R. Roberts. « An improved algorithm for decentralized extrema-finding in circular configurations of processors ». *Communications of the ACM*, 22(5):281–312, 1979.

[COU 94] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems—Concepts and Design, 2nd Ed.* Addison-Wesley Publishers Ltd., 1994.

[FID 91] C. J. Fidge. « Logical Time in Distributed Computing Systems ». *IEEE Computer*, 24(8):28–33, August 1991.

[LAM 78] L. Lamport. « Time, Clocks, and the Ordering of Events in a Distributed System ». *Communications of the ACM*, 21(7):558–565, July 1978.

[LI 89] K. Li and P. Hudak. « Memory Coherence in Shared Virtual Memory Systems ». *ACM Trans. on Computer Systems*, 7(4):321–359, November 1989.

Références

[MAR 88] J. Legatheaux Martins and Y. Berbers. « La désignation dans les systèmes d'exploitation répartis ». *Technique et Science Informatiques*, 7(4):359–372, avril 1988.

[MAT 89] F. Mattern. « Virtual time and global states of distributed systems ». In *Proc. of Int. Workshop on Parallel and Distributed Algorithms, Bonas*, pages 215–226, 1989.

[ORF 95] R. Orfali, D. Harkey, and J. Erwards. *Client/Serveur : guide de survie*. International Thomson Publishing, Paris, 1995.

[RAY 96] R. Raynal and M. Singhal. « Capturing Causality in Distributed Systems ». *IEEE Computer*, 29(2):49–56, February 1996.

[RIF 95] J. M. Rifflet. *La communication sous UNIX : applications réparties*. Ediscience International, 2rd edition, 1995.

[SCH 88] F. Schmuck. « *The use of efficient broadcast in asynchronous distributed systems* ». Tr88-928, Cornell University, 1988.

[TAN 94] A. Tanenbaum. *Systèmes d'exploitation : systèmes centralisés et systèmes distribués*. Interéditions, Paris, 1994.

Sommaire

1. Principes généraux sur les systèmes répartis.
2. Paradigmes et services de communication.
3. Principaux problèmes de coordination et de cohérence de donnée.
4. Conclusion, synthèse.
5. Références.
6. Acronymes.

Acronymes (1)

- **NFS**. Network File System.
- **NIS**. Network Information Service.
- **RPC**. Remote Procedure Call.
- **XDR**. External Data Representation.
- **URL**. Universal Resource Locators.
- **DNS**. Domain Name System.
- **HTML**. Hypertext Markup Language.
- **HTTP**. Hypertext Transfer Protocol.
- **RMI**. Remote Method Invocation.
- **CORBA**. Common Object Request Broker.
- **IDL**. Interface Definition Language.

Acronymes (2)

- **NTP**. Network Time Protocol.
- **UDDI**. Universal Description, Discovery and Integration.
- **SOAP**. Simple Object Access Protocol.
- **WSDL**. Web Services Description Language.
- **XML**. Extensible Markup Language.
- **GIOP**. General Inter-ORB Protocol.