Algorithmique Distribuée
Introduction et Contexte

Laurent PHILIPPE

Master 2 Informatique
UFR des Sciences et Techniques

2013/2014

Introduction

Qu'est que l'algorithmique distribuée ?

De l'algorithmique...

- Solution à un problème de traitement
- Tri
- Graphes
- Synchronisation

...Pour les systèmes distribués

- Introduction de la communication
- Contraintes spécifiques

Exemple

Système distribué

- Une entreprise de voyage (bus)
- 4 antennes réparties géographiquement

Problème

- Les clients réservent leur(s) place(s) dans les agences
- Peuvent annuler leur voyage
- Réservations simultanées pour la(les) place(s)
- Garantir pas de surbooking ni bus sous-utilisé, optimal

Solutions intuitives

- Interroger tous les autres lors d'une réservation
- Informer tous les autres lors d'une annulation

Sommaire

Sommaire

Sommaire

Système Centralisé

Caractérisation

Ensemble de composants (cœurs, processeurs, périphériques) partageant une mémoire et une horloge

Propriétés

- Possibilité de dater de manière non ambiguë les évènements
- Possibilité d'accéder à une donnée commune
- Partage de ressources : processeur, mémoire, disque, périph, ...
- Programmation concurrente synchronisée : sémaphore, mutex
- Exemple : problème de synchronisation en Java multi-thread

Système Distribué

Caractérisation

Ensemble de composants (ordinateurs,téléphones,embarqués,...) autonomes reliés par un réseau de communication, sans partage.

Programmation distribuée

- Pas d'état global : un processus ne peut pas connaître instantanément l'état des autres processus
- Pas d'horloge globale : chaque processus a sa propre horloge
- Echanges : communication et synchronisation, uniquement par échange de messages
- Risque de perte de messages
- Risque de fautes

Systèmes Distribués : avantages / inconvénients

Avantages

- Accès et partage de ressources distantes : distribution permet la mise à disposition de services
  - ressources physiques : imprimantes, espace disque, processeurs, etc
  - ressources logiques : fichiers, données textuelles, audio, images, vidéo
- Répartition géographique : ex : système de réservation avec des centres dans différents lieux
- Puissance de calcul : connexion de machines en réseau permet d'obtenir une puissance de calcul à moindre coût
- Disponibilité : relative indépendance des défaillances des différents sites

Systèmes Distribués : avantages / inconvénients

Inconvénients

- Pas d'horloge globale
- Pas d'état global immédiat accessible à un site
- Fiabilité relative : distribution permet d'introduire de la tolérance aux fautes
- Sécurité relative : architecture distribuée plus difficile à protéger (plusieurs points d'accès aux ressources, évolution dynamique de l'architecture)

Sommaire

Les systèmes de calculs distribués

Les clusters :

- Ensemble de nœuds identiques ayant le même OS connectés par un réseau local high speed (ex : InfiniBand)
- Architecture caractérisée par une certaine homogénéité
- Utilisés pour la programmation parallèle : un même programme est exécuté en parallèle sur plusieurs machines. (SPMD)

Les systèmes de calculs distribués

Les grilles :

- Système profite d'un ensemble de machines qui sont ou ne sont pas dédiées
- Machines peuvent être éloignées
- Architectures hétérogènes : au niveau hardware, système ....
- Calcul, données, applications, etc.
- Types de grilles :
  - Desktop grid
  - Resource Grid
  - Service Grid

Les systèmes d'information distribués

Systèmes orientés données

- Les systèmes à base de transactions : systèmes de gestion de bases de données distribués.
- Les EAI (Enterprise Application Integration) : coordination de différents systèmes
- Systèmes coopératifs
- Midleware : gestion coordonnées de systèmes
- WEB Services : accès à des serveurs distants

Stabilité des systèmes précédents

Propriétés

- Noeuds fixes
- Noeuds fiables : peu de fautes
- Connexion permanente et de haute qualité

Les systèmes mobiles et "pervasifs"

Exemples

- Ordinateurs portables, téléphones, etc.
- Robots, drônes, etc
- Réseaux de capteurs : collecte dans un environnement, surveillance médicale, ...
- Réseaux diffus : domotique, réseaux had-hoc, ...

Instabilité

- Noeuds peuvent apparaître/disparaître
- Pas toujours de connexion réseau
- Mobilité des noeuds
- Systèmes peu traités dans ce cours

Sommaire

Une couche middleware

Définition

Couche d'exécution distribuée entre les systèmes d'exploitation locaux et les applications

Objectifs

Offrir une interface permettant de :

- rendre indépendantes les applications par rapport aux plates-formes
- cacher la distribution aux utilisateurs : offrir une interface la plus proche possible du "centralisé"

Une couche middleware

Services facilitant l'exploitation des ressources

- Datation
- Synchronisation
- Nommage/Localisation
- Courtier
- Sécurité
- Messaging
- Transactions distribuées
- ...

Distribué ou centralisé ?

Centralisé souvent plus facile mais...

- Concentration en un point conduit à la surcharge
- Fragilité face à la tolérance aux pannes
- Pas toujours adapté aux besoins (ex : traitements locaux)

Distribué souvent plus résistant mais...

- Difficulté de mise en œuvre
- N'évite pas tous les risques de surcharge (ex : diffusion systématique de messages)
- Preuve complexe

Distribué ou centralisé ?

Les deux concepts ne sont cependant pas opposés

- Modèles mixtes :
  - Hiérarchique (ex : un maître délègue aux serveurs secondaires)
  - Redondance (ex : serveur de backup)
  - Super-nodes
- Choix dépend des difficultés de réalisation : pragmatisme

Sommaire

Modèle conceptuel de système distribué

Objectifs

- Modèle
- Description statique et comportementale
- Abstraction pour faciliter l'analyse
- Validation et preuve de propriétés

Les éléments de modélisation

- Processus, sites, nœuds : notion de site logique (processus)
- Communication : liens, topologie, protocoles (point à point, diffusion), ...
- Connaissance partielle de chaque site logique

Sommaire

Site logique ou processus

Définitions

- Elément logiciel effectuant une tâche donnée
- tâche : ensemble d'instructions
- instruction : correspond à un évènement local du processus
- évènement local :
  - changement d'état du processeur,
  - émission de message
  - réception de message

Site logique ou processus

Possède

- Un identifiant unique
- Un état local
- Une mémoire locale
- Une horloge locale

Connaît (hypothèse)

- Les processus avec qui il peut communiquer et leur nombre (voisins)
- Les procédures de communication
- Eventuellemnt l'état approximatif des autres processus

Sommaire

La communication

Topologie

La topologie d'un réseau décrit la façon dont les processus peuvent communiquer entre eux.

Exemple de topologies

- anneau uni ou bi-directionnel
- réseau complet
- arbre
- grille

Protocoles de communication

- point à point
- diffusion

Communication synchrone/asynchrone

Communication synchrone

- Entre les processus P_i et P_j se fait par rendez-vous : le premier processus qui atteint un point de communication attend l'autre
- Synchronisation entre l'émetteur et le récepteur

Communication asynchrone

- Le processus P_i envoie un message et continue son exécution sans se préoccuper si P_j l'a reçu
- Délais de communication arbitraires mais finis
- On ne sait pas quand un message émis sera reçu
- Cas particulier de la perte de message : algorithmes tolérants aux pannes

Sommaire

Modèle de système distribué

Graphe de processus

- Graphe : $G(\mathbb{P}, \mathbb{C})$
- Sommets = sites logiques, $\mathbb{P}$ ensemble de processus
- Arcs = liaisons de communication entre les sites logiques, $\mathbb{C}$ ensemble de liens
- Arc entre P_i et P_j signifie que P_i et P_j peuvent communiquer sans intermédiare

Modèle d'exécution distribuée

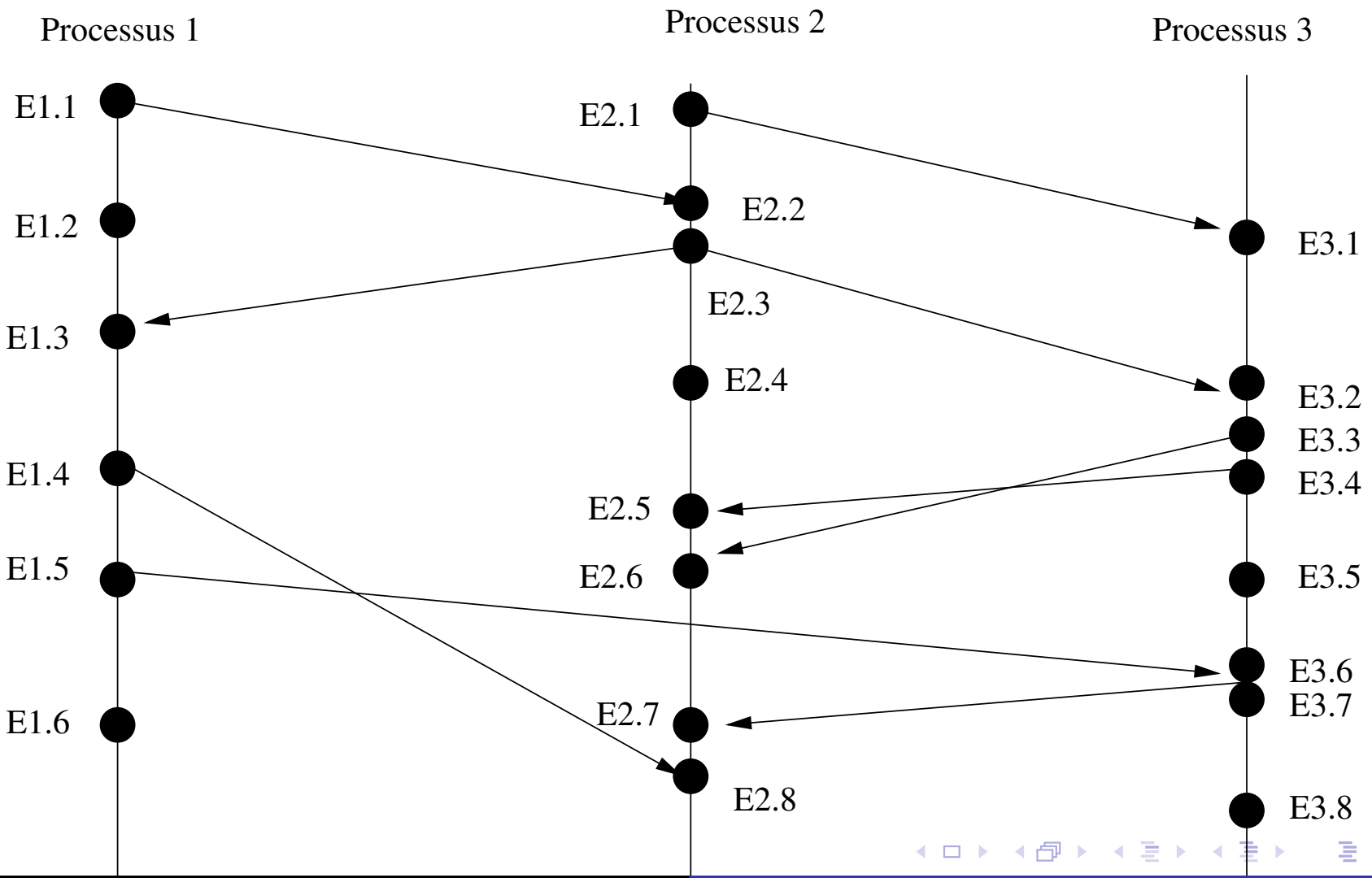
Chronogramme

- Représentation graphique
- Ordre des évènements
- Déroulement chronologique sur chaque site
- Représentation simple mais limitée

Évènements

- Emission de message
- Réception de message
- Evènement interne

Exemple de chronogramme



Modèle d'exécution distribuée

Modèle des évènements

- Pour une exécution $\mathcal{C}$ (calcul) distribuée d'un ensemble $\mathbb{P}$ de processus
- L'ensemble des évènements est noté E_v,
- L'exécution locale d'un processus p est composée d'une séquence $c_p^0, c_p^1, c_p^2, ...$ d'états du processus, où c_p^0 est l'état initial.
- La transition du l'état c_p^{i-1} à l'état c_p^i du processus p est marquée par l'occurence de l'évènement e_p^i
- On définit $E_v = \bigcup_{p \in \mathbb{P}} \{e_p^0, e_p^1, e_p^2, ...\}$

Modèle d'exécution distribuée

Modèle des communications

- Un évènement peut-être interne, une émission ou une réception
- L'état c_p^i contient donc l'ensemble des variables du processus dans l'état i et l'historique complète des communications :
  - La liste $sent_{pq}^i$ des messages envoyés de p à q parmi les évènements e_p^0 à e_p^i,
  - la liste $rcvd_{pq}^i$ des messages que q à reçu de p parmi les évènements e_p^0 à e_p^i,

Restrictions du modèle de système distribué

- Nombre de processus est constant
- Topologie de communication est fixe
- Seules interactions possibles : échange de messages
- Aucun processus n'est isolé

Sommaire

L'algorithmique distribuée

Qu'est ce que c'est ?

- Algorithmique
- Point de vue modélisé
- Spécifique aux problèmes soulevés par les systèmes distribués

Caractéristiques

- Programmes + Communication par messages
- Pas de donnée partagée : chaque programme défini ses propres données
- Exécution : déroulement en parallèle des programmes, synchronisation/échanges sur les messages
- Preuve

Modèle algorithmique

Algorithme des processus

- Tous les processus exécutent le même algorithme
- Décomposé en :
  - Données, les données locales aux processus, valeur d'initialisation
  - Messages, les messages utilisés (envoyés/reçus) par le processus, peuvent contenir des données si besoin
  - Règles, les fonctions du processus. Composées en général de :
    - Règle d'intialisation, processus initiateur
    - Règle prinicipale, liée à la fonction de l'algorithme
    - Règles de réception des messages, sur le mode message

  En général on fait l'hypothèse du traitement atomique, au moins sur les règles de réception des messages

Modèle algorithmique

Modèle de communication

- Différents types de messages (ACK, REQ, etc.)
- envoyer : `Envoie TYPE(parametres) à destinataire`
- recevoir : `Reçoit TYPE(parametres) de emetteur`
- Définition de règles associées aux réceptions

Propriétés

- Synchronisation dépend des proriétés des fonctions de communication
- Peuvent être définies spécifiquement (précisé)
- Mais généralement asynchrone : envoi du message sans attendre sa réception

Exemple : diffusion dans un anneau

Variables pour chaque processus P_i :
$Succ_i$: Successeur de P_i dans l'anneau

Messages utilisés :
BROADC(P, Msg) : Message de diffusion

Règle 1 : P_i *demande de diffuser Msg* sur l'anneau
début
  Envoie BROADC(i, Msg) à $Succ_i$
fin

Règle 2 : P_i *reçoit BROADC(init, Msg) de* P_j
début
  if *init* !=*i* **then**
    Envoie BROADC(*init*, Msg) à $Succ_i$
fin

Sommaire

Exemple simple

Accès partagé à un garage

- N portes
- P places de parking

Problème

- Limite du nombre max de voitures
- Chaque porte d'entrée dispose seulement de données locales
- Comment savoir si une voiture peut sortir ?
- Comment savoir si une voiture peut entrer ?

Accès partagé à un garage

Contraintes/Hypothèses

- Pas de données partagées
- Pas d'horloge commune
- Échanges entre portes uniquement basés sur les messages
- On suppose que toutes les portes peuvent communiquer directement entre elles
- Une voiture qui demande à entrer/sortir entre/sort effectivement

Exercice

Proposer plusieurs solutions permettant de garantir qu'aucune voiture ne rentre sans avoir de place.

Accès partagé à un garage

Solutions

1. Une porte centralise la valeur
2. Un quota est attribué à chaque porte
3. Les portes qui n'ont plus de quota interrogent les autres pour savoir si une place s'est libérée
4. Les portes communiquent entre elles pour gérer le compteur en commun
5. Chacun travaille indépendemment, collaboration seulement si le nombre de places passe sous un seuil
6. Et encore bien d'autres solutions...

Accès partagé à un garage

Exercice

1. Ecrire les algorithmes des portes pour l'entrée et la sortie d'une voiture dans le cas centralisé.
2. Donner la preuve de cet algorithme

Contraintes

- Les portes secondaires demandent à la porte principale
- Pas de mise en attente des voitures : plus de place = refus d'accès
- Le résultat attendu est une variable `ACCORD` ou `REFUS`

Solution - 1 : porte principale (maître)

Variables la porte P_m : int compteur initialisé a P

Messages utilisés : ? ?

Règle 1 : *Une voiture demande d'entrer*

```
début
    if compteur == 0 then
        return REFUS
    else
        compteur = compteur - 1
        return ACCORD
fin
```

Règle 2 : *Une voiture demande de sortir*

```
début
    compteur = compteur + 1
fin
```

Solution - 1 : portes secondaires (esclaves)

Demandent à la porte principale :

- message *ACCES*, de demande
- message *RETOUR*(*rep*), qui contient la réponse
- message *SORTIE* pour informer qu'une voiture sort

Solution - 1 : portes secondaires (esclaves)

Variables la porte P_i : m identificateur de la porte maître

Messages utilisés :
ACCES : demande d'accès pour une voiture
RETOUR(*rep*) : réponse *rep* du maître
SORTIE : informe la porte maître de la sortie d'une voiture

Règle 1 : *Une voiture demande d'entrer*
début
| Envoie *ACCES* à m
fin

Règle 2 : *Réception du message RETOUR*(*rep*) *du maître* **début**
| return rep
fin

Règle 3 : *Une voiture demande de sortir*
début
| compteur = compteur + 1
fin

Solution - 1 : porte principale (maître)

Variables la porte P_m : int compteur initialisé a P

Messages utilisés :
ACCES : demande d'accès pour une voiture
RETOUR(*rep*) : réponse *rep* du maître
SORTIE : informe la porte maître de la sortie d'une voiture

Règle 1 : *Une voiture demande d'entrer*

Règle 2 : *Une voiture demande à sortir*

Règle 3 : *Reçoit un message ACCES de* P_i
début
  if *compteur* == *0* **then** envoie *RETOUR*(*REFUS*) à P_i
  else
    compteur = compteur - 1 ; *RETOUR*(*ACCORD*) à P_i
fin

Règle 4 : *Reçoit un message SORTIE*
début
  compteur = compteur + 1
fin

Solution - 1

Preuve

Montrer la proposition : aucune voiture entre dans le parking si il n'y a pas de place disponible

Idée

Repose sur une autre assertion : le nombre compteur est le compte exact du nombre de voitures dans le parking :

- Quand une voiture entre, quelque soit la porte utilisée, le compteur est décrémenté
- Quand une voiture sort, quelque soit la porte utilisée, le compteur est incrémenté

Solution - 1

Problèmes

- Synchronisation chez la porte principale :
  - concurence entre les fonctions locales et distantes,
  - concurence entre les fonctions distantes
- Centralisé : panne de la porte principale entraîne une panne du parking

Accès partagé à un garage

Exercice

1. Écrire les algorithmes des portes pour l'entrée et la sortie d'une voiture dans le cas où un quota est attribué à chaque porte
2. Donner la preuve de cet algorithme

Contraintes

- Au début chaque porte se voit attribuer le même quota
- On suppose que le nombre de places est divisible par le nombre de portes
- Pas de mise en attente des voitures : plus de place = refus d'accès
- Le résultat attendu est une variable `ACCORD` ou `REFUS`

Solution - 2 : quotas d'entrées par porte

Variables la porte P_i : int compteur initialisé à $\frac{P}{N}$

Messages utilisés :

Règle 1 : *Une voiture demande d'entrer*
début
  if *compteur == 0* **then**
    return REFUS
  else
    compteur = compteur - 1
    return ACCORD
fin

Règle 2 : *Une voiture demande à sortir*
début
  compteur = compteur + 1
fin

Solution - 2

Preuve

Montrer la proposition : aucune voiture n'entre dans le parking si il n'y a pas de place disponible

Idée

Bien sûr cet algorithme ne permet pas de garantir cette proposition, il suffit pour cela de donner un exemple où cela ne marche pas

- A partir de l'état intial, $\frac{P}{N} + 1$ voitures arrivent à la porte i
- Aucune voiture sort par cette porte
- Aucune voiture n'entre par une autre porte
- La dernière voiture qui arrive à la porte i ne peut pas rentrer alors qu'il reste des places

Solution - 2

Problèmes

- Ne permet pas d'utiliser toutes les places
- Toutes les voitures sortent par la même porte
- Une porte tombe en panne

Accès partagé à un garage

Exercice

1. Solutions 2 + lorsque le quota est épuisé, les portes s'interrogent entre elles pour savoir si une place s'est libérée
2. Donner la preuve de cet algorithme

Contraintes

- Au début chaque porte se voit attribuer le même quota
- On suppose que le nombre de places est divisible par le nombre de portes
- Pas de mise en attente des voitures : plus de place = refus d'accès
- Le résultat attendu est une variable `ACCORD` ou `REFUS`

Solution - 4

Les portes communiquent pour gérer le compteur en commun

Quelques pistes :

- Mise en place d'un jeton :
  - Le jeton porte le nombre de places : doit être rapide si grand nombre de portes
  - le jeton distribue des places libres aux portes d'entrée
- Consensus
- Maître tournant : le maître peut modifier la valeur, demande à devenir maître, enregistre à qui a donné la maîtrise

Exercices

A faire

Solution - 4 : anneau

A faire

Solution - 5 : consensu

Sommaire

Complexité d'un algorithme distribué

Deux points de vue sur la complexité

1. Temps local au processus
   - Temps de l'exécution d'un calcul local
   - Temps de transfert de message
2. Temps entre processus
   - Nombre de messages
   - Nombre de processus impliqués

Choix de complexité dépend des propriétés du système : performances réseau, topologie, etc.

Propriétés d'un algorithme distribué

Propriété de sûreté

- Un état catastrophique ne sera jamais atteint
- Exemple : accès en exclusion mutuelle où une section critique est accessible par au plus un processus

Propriété de vivacité

- Un état de satisfaction sera fatalement atteint
- Exemple : toute demande d'accès à la section critique sera satisfaite ou encore tout message émis sera reçu

Propriétés d'un algorithme distribué

Propriété d'équité

- Garantir que les processus ont tous les mêmes chances d'accès à une ressource
- Exemple : il n'y a pas un processus qui accède plus à une section critique

Propriété de terminaison

- L'algorithme distribué se termine ou conduit à une décision
- Exemple : cas de l'élection d'un maître

Propriété de ponctualité

- Des contraintes temporelles peuvent être fixées
- Exemple : une échéance de remise de messages au plus tard

Sommaire

Objectifs du cours

- Première approche : comprendre les problèmes
- Connaître les algorithmes
- Connaître des solutions aux problèmes
- Savoir les appliquer à différents problèmes

Sommaire

Références

- Cours de Gérard Padiou (ENSEEIHT Toulouse)
- Cours de Eric Cariou (Université de Pau)
- Livres :
  - A.Tannenbaum "Distributed Systems" Prentice Education Ed.
  - A.D. Kshemkalyani, M. Singhal, “Distributed Computing : Principles, Algorithms, and Systems”, Cambridge University Press