UPMC
SORBONNE UNIVERSITÉS

AR – 4I403

Algorithmique répartie
2016-2017

Luciana Arantes, Swan Dubois, Claude Dutheillet, Franck Petit

Prenom.Nom@lip6.fr

Responsable : Franck Petit

Organisation

11 semaines :

Cours, TD, TME, mini-projet

TME : MPI et Promela (spin)

Projet : pair à pair

Attention : Dernier cours le **mardi 2 mai**

Evaluation:

premier écrit : 30%

deuxième écrit : 50 %

projet : 20%

Programme de l'UE

- Terminologie, topologie
- Temps, causalité, horloge logique
- Algorithmes à Vagues
- Exclusion mutuelle
- Présentation de SPIN
- Election
- Introduction aux systèmes à large échelle (P2P)
- Terminaison
- Etat Global

Bibliographie

- Gerard Tel, *Introduction to Distributed Algorithms*, Cambridge University Press, 1994, 2000 (2ème edition).
- Nancy Lynch, *Distributed Algorithms*, Morgan Kaufmann, 1996.
- Michel Raynal, *Synchronisation et état global dans les systèmes répartis*, Eyrolles, 1992
- H. Attiya and J. Welch, *Distributed Computing. Fundamentals*, Simulations and Advanced Topics, McGraw-Hill, 1998.

Introduction Algorithmique répartie - Plan

- **Système réparti**
  - ➔ mémoire partagée *vs* échange de messages
- **Topologie des systèmes répartis**
- **Modèles de Fautes et Modèles Temporels**
- **Problèmes inhérents à la répartition**
- **Evaluation et vérification d'un algorithme réparti**

Système réparti

Qu'est-ce qu'un système réparti ?

Ensemble interconnecté d'entités autonomes

qui communiquent via un médium de communication (G. Tel)

Entités :

ordinateurs, processeurs, processus, routeurs, bases de données, PDA, robots mobiles…

Autonomes :

chacune des entités possède son propre contrôle.

Interconnexion :

capacité à échanger de l'information : canaux de communication ou mémoire partagée.

Caractérisation d'un calcul réparti

Non séquentiel :

Deux instructions peuvent être exécutées simultanément

Non centralisé :

Les paramètres décrivant l'état du système sont répartis

Non déterministe :

Deux actions concurrentes peuvent être exécutées dans n'importe quel ordre.

Le comportement d'une entité peut dépendre de ses interactions avec les autres entités.

Buts d'un système réparti

But technique : mise en commun des **ressources** matérielles de plusieurs machines

- processeur et mémoire : + de capacité de calcul
- disques : + de capacité de stockage
- imprimantes
- …

→ Factorisation des coûts

→ Partage de charge

→ Tolérance aux pannes

But fonctionnel : mise en commun d'**informations** entre plusieurs utilisateurs ou systèmes

- fichiers ou bases de données
- événements ou alarmes
- …

→ Travail coopératif entre utilisateurs

→ Automatisation de chaînes de traitement

Classification des applications réparties

Deux classes d'applications :

Processus coopérants :

Les processus interagissent via des mémoires ou des variables partagées.

➔ gérer les conflits d'accès aux ressources communes (exclusion mutuelle)

Processus communicants :

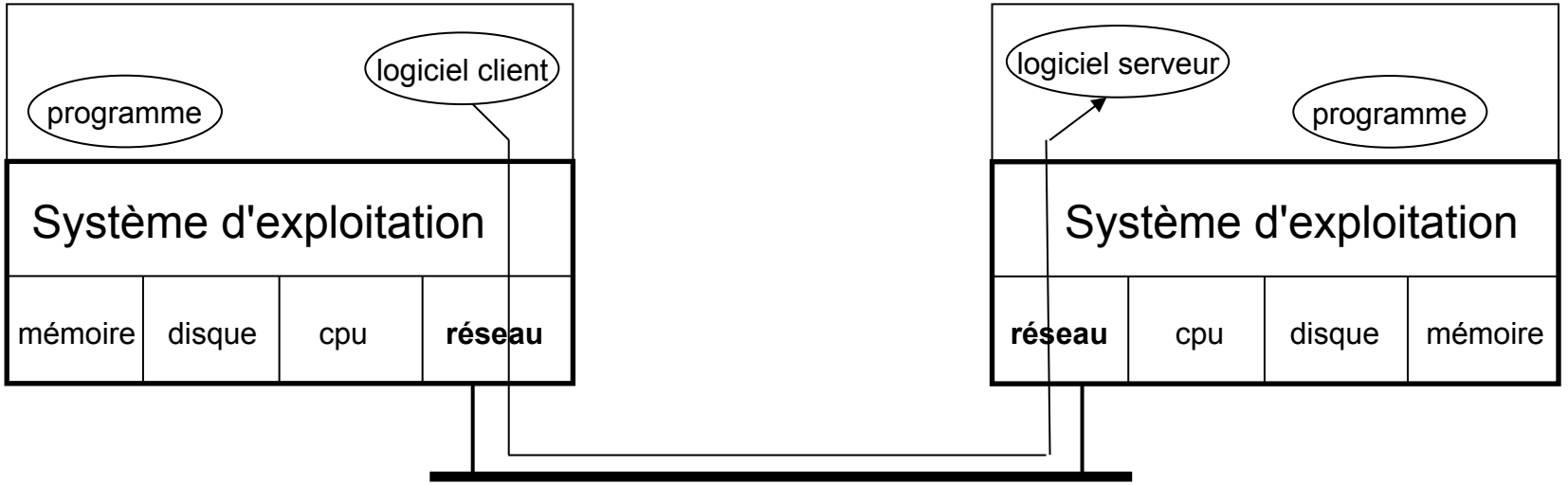
Les processus s'échangent des messages par l'intermédiaire de canaux.

➔ gérer l'échange de la connaissance

Exemples

- Système étendu (client/serveur)
- Système à architecture répartie
- Système d'exploitation réparti

Système étendu (client–serveur)



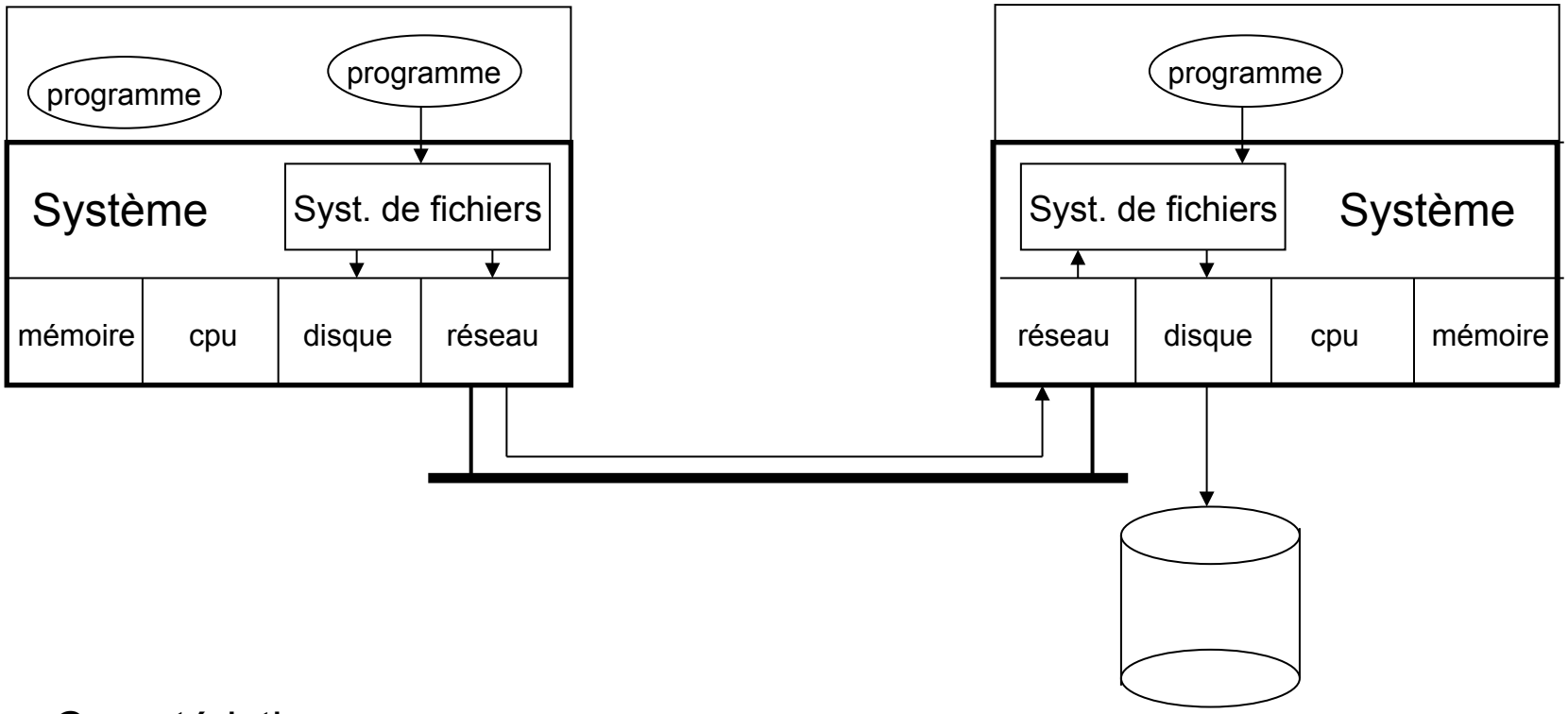
Caractéristiques :

- **un** SE **indépendant** par machine
- un logiciel **explicite** de communication (login, telnet, ftp, rcp, …)

Exemples :

Unix/Linux, MacOs, Windows…

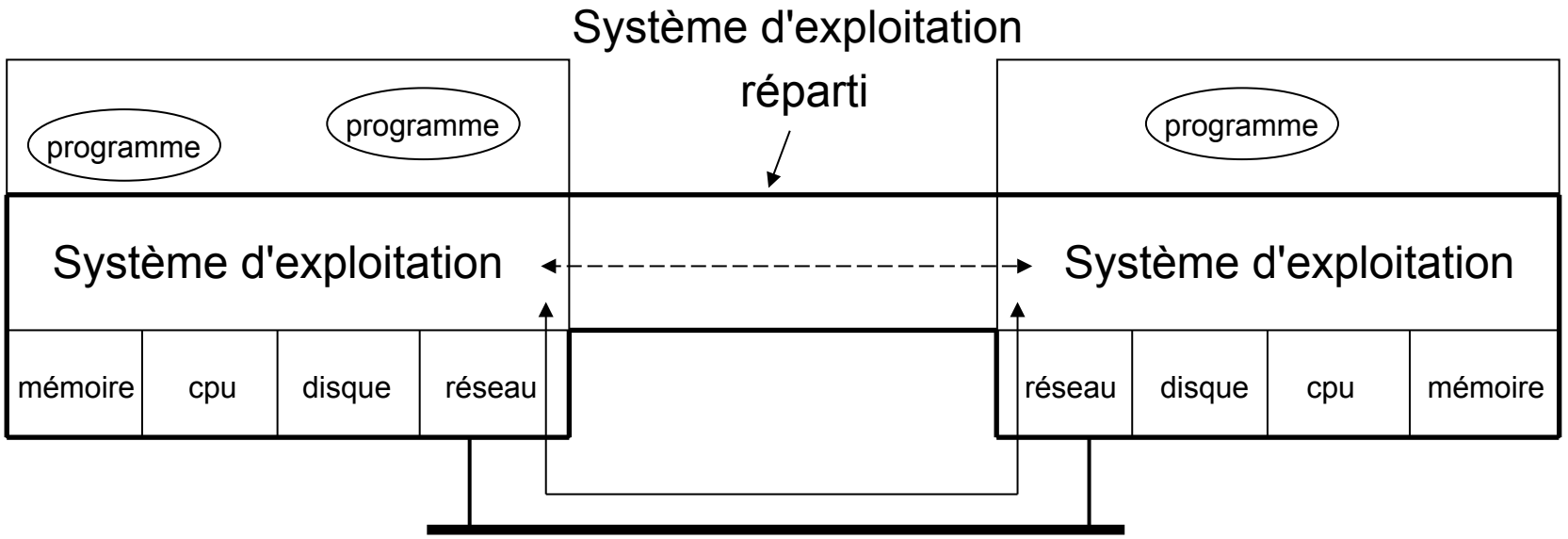
Système à architecture répartie



Caractéristiques :

- **un** SE **indépendant** par machine
- un accès **transparent** à certaines ressources distantes

Système d'exploitation réparti



Caractéristiques :

- les SE locaux **coopèrent** pour rendre leurs services (cpu, memoire, disque)
- et donner l'illusion d'une **machine virtuelle unique** composée des **ressources de tous les sites**

Exemples :

ChorusOS (INRIA – 80')

V-System (Stanford U. – 84)

Mach (Carnegie Melon U. – 86)

Amoeba (Amsterdam U. – 90)

Canaux de communication

Risque de déséquencement des messages

- ➔ P envoie M_1 puis M_2 à Q. Q reçoit M_2 puis M_1
- ➔ Pas de déséquencement = canal FIFO

Risque de perte de messages

- ➔ Un message émis n'est jamais reçu
- ➔ Communications fiables : tout message émis est reçu exactement une fois
  - au bout d'un temps arbitraire, mais fini.
  - sans détérioration.

Certains canaux peuvent avoir une capacité bornée

On se place ici au niveau logique et pas au niveau physique !

Topologies de systèmes répartis

Topologie des systèmes

Représentation sous forme de graphe :

- ➔ nœuds = processus
- ➔ arêtes = canaux de communication
  - Canal unidirectionnel (dirigé) de p vers q = arc de p vers q
  - canal bidirectionnel entre p et q = arête entre p et q

Caractéristiques du graphe

- ➔ connexe : chaque paire de nœuds est reliée par un chemin.
- ➔ fortement connexe : graphe orienté, où il existe un chemin entre chaque paire de nœuds, en respectant le sens des arcs.
- ➔ incomplet : tous les processus ne communiquent pas deux à deux directement.

La topologie du graphe peut être un critère pour l'existence d'une solution répartie à un problème

Paramètres définis sur le graphe

Distance entre deux nœuds :

Longueur du plus court chemin entre ces deux nœuds

Diamètre du graphe :

La plus longue des *distances* entre deux nœuds du graphe

Degré d'un nœud :

Nombre de voisins du nœud

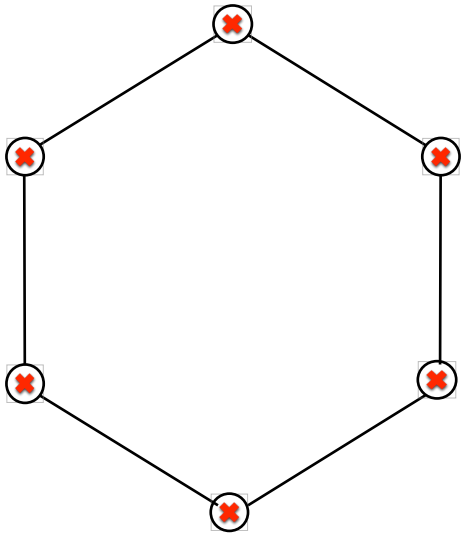
Ces paramètres peuvent être utiles dans

le calcul de la complexité d'une application répartie

Topologies particulières (1)

Anneau unidirectionnel:

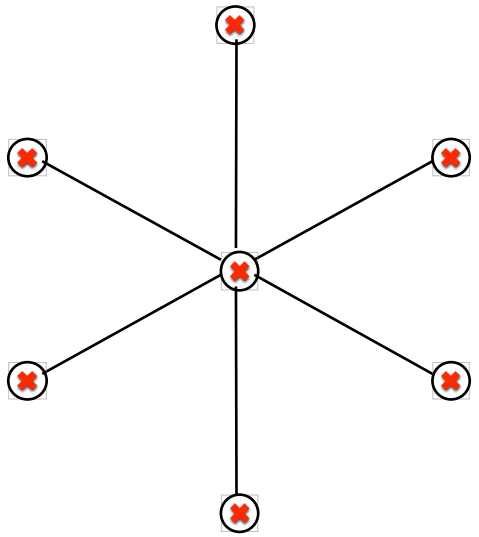
- N nœuds
- N arêtes
- Tous les nœuds sont de degré 2 (ont 2 voisins)
- Diamètre : n/2



Topologies particulières (2)

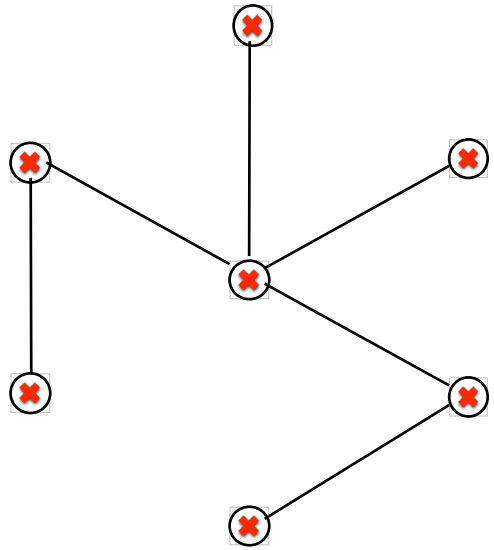
Etoile :

- N nœuds
- (N -1) arêtes
- Tous les nœuds sont de degré 1, sauf le nœud central qui est de degré N-1.
- Diamètre : 2



Arbre :

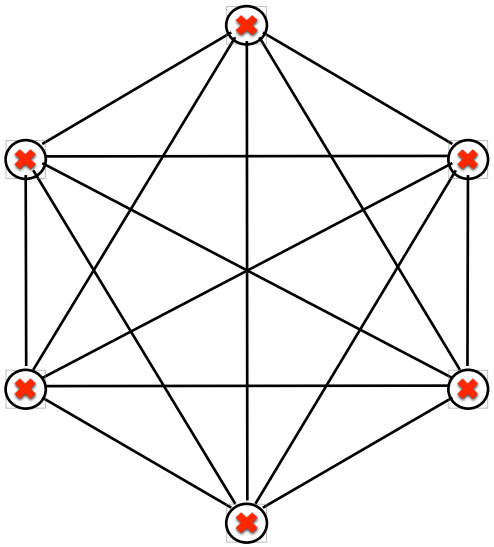
- N nœuds
- (N -1) arêtes
- Pas de cycle



Topologies particulières (3)

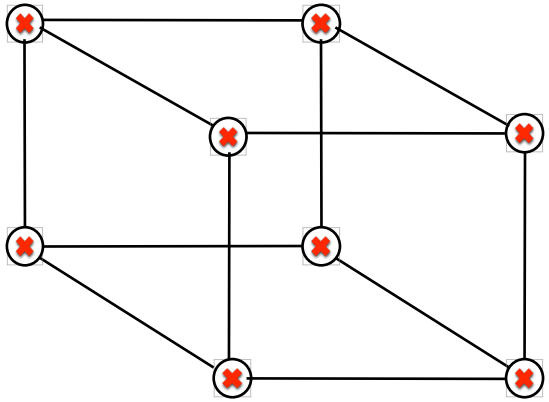
Clique (graphe complet) :

- N nœuds
- Tous les nœuds sont reliés deux à deux par une liaison
- n*(n-1) /2 arcs
- Diamètre : 1



Hypercube :

- N = 2^n nœuds
- Chaque numéro de nœud est codé sur n bits
- Il existe une liaison entre les nœuds dont les numéros ne diffèrent que d'un bit
- Nombre de voisins constant



Topologie physique et topologie logique

Topologie physique :

➔ Câblage du réseau

Topologie logique (structure de contrôle) :

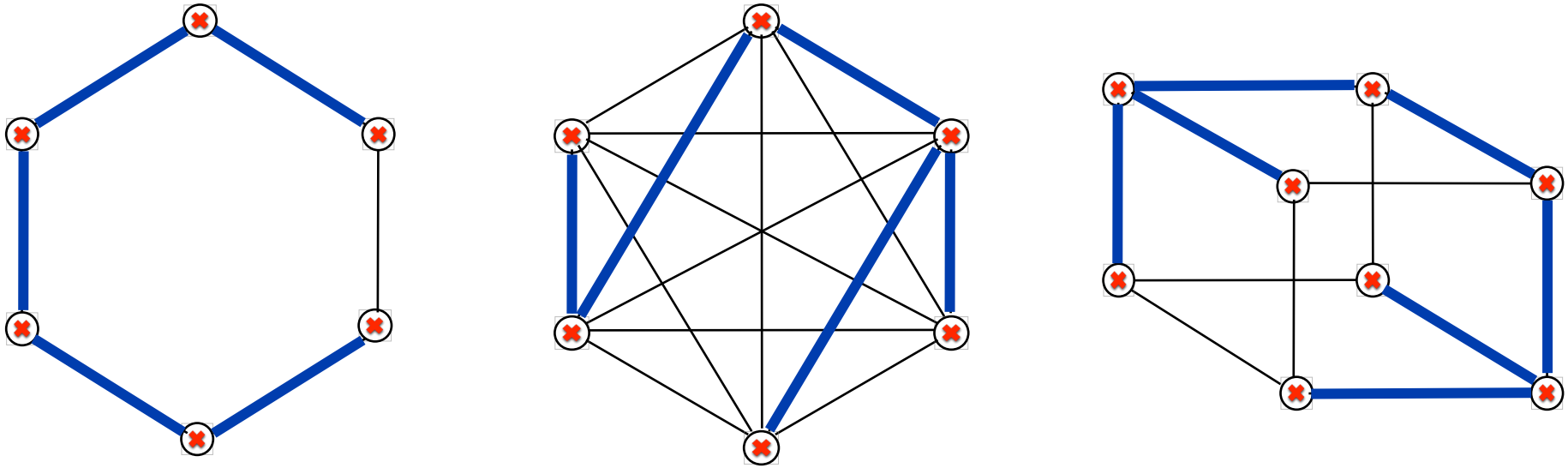
➔ Manière dont la communication entre les sites est organisée

La topologie logique et la topologie physique peuvent différer.

➔ Topologie (ou structure) ***couvrante*** : certaines liaisons physiques ne sont pas utilisées pour la communication entre les sites.

Structure de contrôle en arbre

Sur une topologie physique connexe quelconque, on peut TOUJOURS implanter une structure de contrôle en arbre.

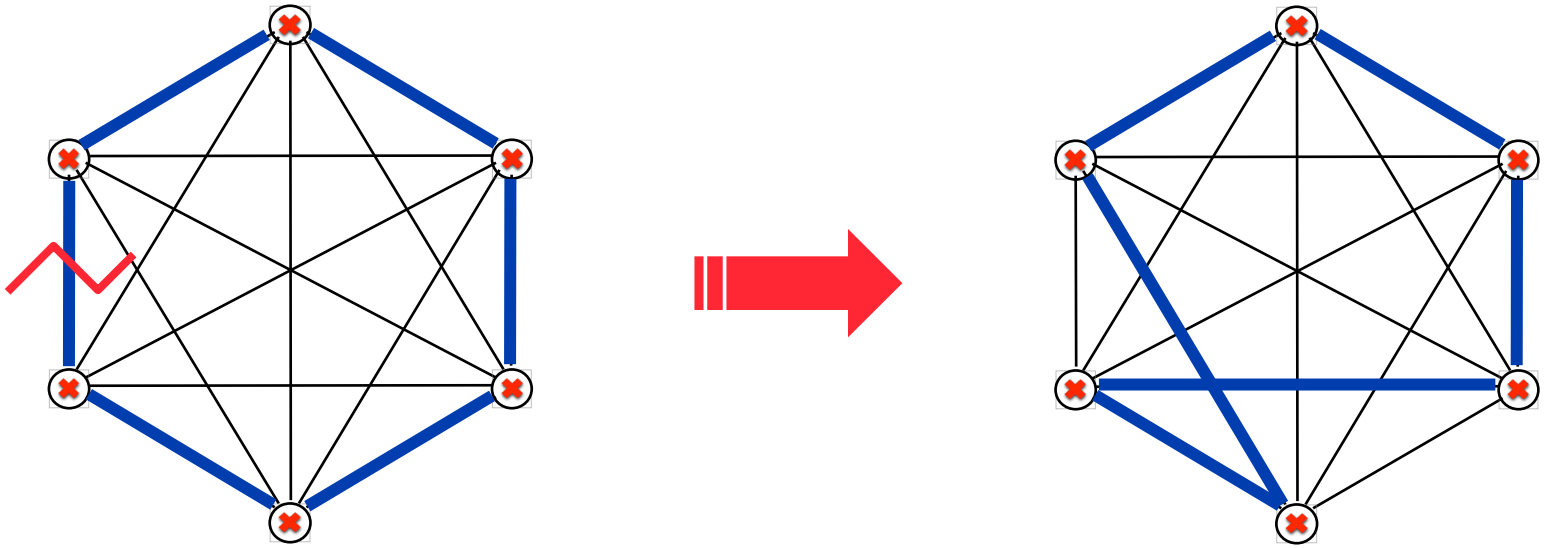


Un ***arbre*** est un **graphe connexe sans cycle.**

Exemple d'utilisation : Diffusion d'information

Reconfiguration d'un anneau

Si une liaison de l'anneau logique tombe en panne, on peut reconstruire un anneau en passant par une autre liaison



Reconfiguration dynamique $\Rightarrow$ coopération des processus pour reconstruire l'anneau

Modèles de fautes et modèles temporels

Modèles de fautes

Origines des fautes

- fautes logicielles (de conception ou de programmation)
  - quasi-déterministes, même si parfois conditions déclenchantes rares
  - très difficiles à traiter à l'exécution : augmenter la couverture des tests
- fautes matérielles (ou plus généralement système)
  - non déterministes, transitoires
  - *corrigées* par point de reprise ou *masquées* par réplication
- piratage
  - affecte durablement un sous-ensemble de machines
  - masqué par réplication

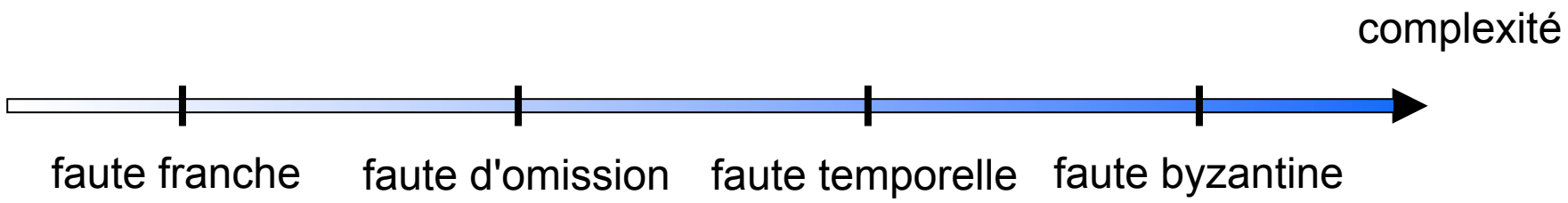
Composants impactés

- processus
- calculateurs
- canaux de communication

Modèles de fautes

Classification des fautes

- faute franche : arrêt définitif du composant, qui ne répond plus ou ne transmet plus
- faute d'omission : un résultat ou un message n'est transitoirement pas délivré
- faute temporelle : un résultat ou un message est délivré trop tard ou trop tôt
- faute byzantine : inclut tous les types de fautes, y compris le fait de délivrer un résultat ou un message erroné (intentionnellement ou non)



Modèles temporels

Constat

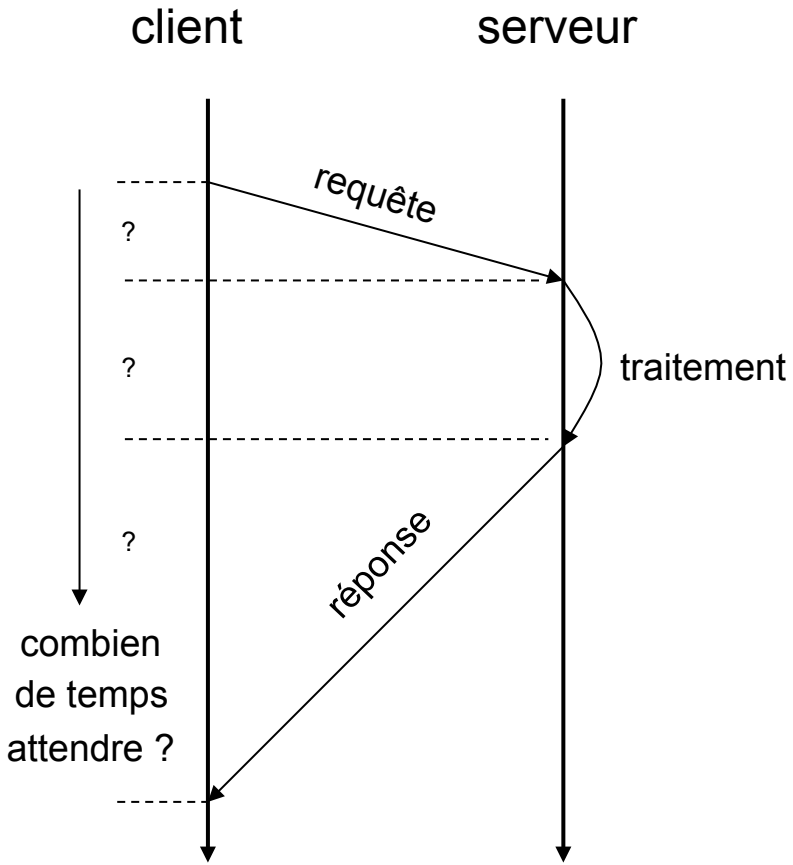
- vitesses des processus différentes
- délais de transmission variables

Problème

- ne pas attendre un résultat qui ne viendra pas (suite à une faute)
- combien de temps attendre avant de reprendre ou déclarer l'échec ?

Démarche

- élaborer des modèles temporels
- dont on puisse tirer des propriétés



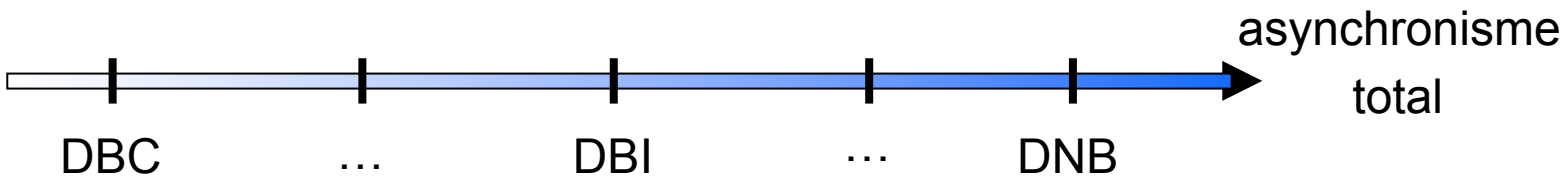
Modèles temporels (2)

Modèle temporel = hypothèses sur :
- délais de transmission des messages
- écart entre les vitesses des processus

système synchrone : Modèle **D**élais/écarts **B**ornés **C**onnus (DBC)
- permet la détection **parfaite** de faute

système partiellement synchrone : Modèle **D**élais/écarts **B**ornés **I**nconnus (DBI)

système asynchrone : Modèle **D**élais/écarts **N**on **B**ornés (DNB)

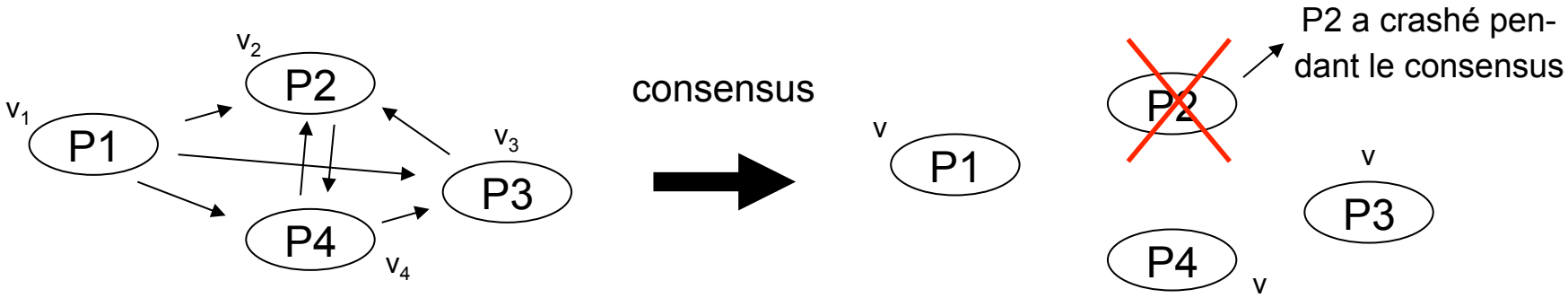


Modèles temporels (3)

Résultat fondamental :

Fischer, Lynch et Paterson 85 : le problème du consensus **ne peut être résolu** de façon déterministe dans un système **asynchrone** dans lequel---ne serait-ce qu'---**une** faute **franche** puisse se produire

Problème du consensus : N processus se concertent pour décider d'une valeur commune, chaque processus proposant sa valeur initiale v_i.



Spécification formelle du consensus :

- terminaison : tout processus correct finit par décider
- accord : deux processus ne peuvent décider différemment
- intégrité : un processus décide au plus une fois
- validité : si v est la valeur décidée, alors v est une des v_i

Notre modèle

Ensemble de processus séquentiels indépendants

➔ Chaque processus n'exécute qu'une seule action à la fois

Communication par échange de messages

➔ Aucune mémoire partagée

➔ Les entrées des processus sont les messages reçus, les sorties sont les messages émis

Système asynchrone (souvent considéré) :

➔ Asynchronisme des communications

- Aucune hypothèse sur les temps d'acheminement des messages (pas de borne supérieure)

➔Asynchronisme des traitements

- Aucune hypothèse temporelle sur l'évolution des processus

Pas d'horloge commune

Problèmes inhérents à la répartition

Critères d'évaluation

Pourquoi existe-t-il plusieurs solutions à un problème de répartition ?

Parce qu'elles ne sont pas forcément comparables.

Les solutions peuvent diverger par

- **les hypothèses qu'elles font sur :**
  - ➔ le système de communication
  - ➔ la connaissance qu'un processus doit avoir de son environnement
  - ➔ la connaissance qu'un processus doit avoir de l'état des autres processus
- **leur capacité à tolérer les pannes**

Le système de communication

Un algorithme réparti fait souvent des hypothèse sur

La topologie du système de communication

Les propriétés comportementales de ce système :

- ➔ fiabilité des liaisons
- ➔ modèles de synchronisme (temporel)
- ➔ possibilité de déséquencement des messages

Moins un algorithme fait d'hypothèses, plus il est général

MAIS…

Un algorithme défini pour un environnement particulier peut être le plus efficace dans cet environnement

Connaissance de l'environnement

Quelle est l'information connue par chaque processus **AVANT** le lancement de l'algorithme ?

En général, un processus possède des informations sur

➔ **la topologie du réseau**

- Nombre de sites
- Diamètre
- Topologie complète

➔ **l'identité des processus**

- Les noms des sites sont uniques
- Chaque site connaît son propre nom
- Un site peut ne pas connaître l'identité de ses voisins, mais il connaît au moins leur existence

Connaissance de l'état des autres

L'état global de l'application n'est pas connu

Un processus ne peut pas prendre une décision sans échanger de l'information avec les autres

Plus la quantité d'information nécessaire à la décision est importante, plus il est difficile de parvenir à cette décision

- ➔ Nombreux échanges de messages
- ➔ Instabilité de l'information obtenue

Résistance aux pannes

Les défaillances peuvent se produire

- ➔ au niveau des processus
- ➔ au niveau des canaux de communication

La capacité de résistance de l'application dépend

- ➔ **du type de défaillance observée**
  - Perte de la fiabilité de la liaison
  - Panne franche de processus (arrêt)
  - Défaillances byzantines
- ➔ **du degré de symétrie de l'application**
  - Aucune symétrie
  - Symétrie de texte (comportement dépend de l'identité)
  - Symétrie totale

Types de Problèmes

Réaliser une opération qui n'existe pas sans répartition ou concurrence de plusieurs processus

- ➔ Routage
- ➔ Exclusion mutuelle
- ➔ Election

Réaliser une opération qui est compliquée parce que l'application est répartie

- ➔ Obtention de l'état du système
- ➔ Détection de la terminaison de l'application

Routage

Calculer le chemin « optimal » sur lequel le processus i envoie ses messages au processus j.

Problème :

- ➔ i ne connaît du réseau que ses voisins
- ➔ i doit trouver quel est le voisin « le plus adapté » pour transmettre ses messages à j

Solution :

- ➔ calcul du plus court chemin dans un graphe en cas de communication point-à-point
- ➔ calcul d'un arbre de recouvrement de poids minimal pour une diffusion

Exclusion mutuelle

Ne pas autoriser les accès simultanés à une ressource pour préserver sa cohérence.

Problème :

si les processus sont géographiquement distants, on ne peut plus utiliser de mécanisme de type sémaphore centralisé.

Solutions :

- ➔ utiliser un jeton qui fait office d'autorisation d'accès
  - Mécanisme de régénération du jeton en cas de perte
- ➔ demander l'autorisation pour accéder à la ressource à tous les autres sites
  - Ne passe pas à l'échelle
- ➔ gérer une file d'attente (centralisée ou distribuée) des requêtes
  - Fragilité d'une file centralisée
  - Difficulté à gérer la cohérence d'une file distribuée

Election

Dans certaines applications, un processus joue un rôle particulier.

Comment choisir ce processus initialement ?
Comment le remplacer s'il tombe en panne ?

Solution :

- ➔ Définir un critère d'élection unique et qui puisse toujours être satisfait
- ➔ Faire participer tous les processus à l'élection
- ➔ Gérer la communication de sorte qu'à la fin de l'élection, un processus est dans l'état ELU, tous les autres sont dans l'état BATTU

Contrainte sur l'application :

- ➔ Tous les processus éligibles doivent posséder la partie de code correspondant au rôle particulier

Calcul d'état global

Qu'est-ce qui définit l'état du système ?

- ➔ L'état de chacun des processus à un instant donné
- ➔ L'état à ce même instant de tous les canaux de communication

Pourquoi cet état est-il difficile à obtenir ?

- ➔ Les informations sont réparties géographiquement
- ➔ Il n'y a pas d'horloge globale

Quelle est la solution ?

- ➔ Collecter « intelligemment » les informations auprès de chaque processus
  - Synchroniser la prise d'information
  - Ne pas perturber l'application observée

Détection de la terminaison

Comment être certain qu'à un instant *t* tous les processus sont *définitivement* inactifs ?

L'inactivité des processus correspond-elle à la terminaison correcte de l'application ?

- ➔Détecter qu'un processus inactif ne redeviendra jamais actif
- ➔Détecter que les canaux sont vides

Difficulté :

un processus ne peut pas toujours savoir à partir de son information locale si l'application est terminée ou non.

Solution :

coordonner la prise d'information

Evaluation et vérification d'un algorithme réparti

Critères d'évaluation

La complexité en nombre d'opérations est peu significative

- ➔ Les opérations sont exécutées par des sites différents
- ➔ Tous les sites n'exécutent pas le même nombre d'opérations

Complexité en messages :

- ➔ Nombre total de messages échangés au cours de l'exécution
- ➔ Critère biaisé si les tailles de messages sont très différentes
  - Complexité en nombre de bits d'information échangés

<u>Complexité en temps</u> :

- ➔ Comment définir une complexité en temps lorsque le système ne possède pas d'horloge globale ?

Complexité en temps

En l'absence d'horloge globale, utilisation d'une notion idéalisée du temps :

- ➔ le temps d'exécution d'un pas de calcul est nul
- ➔ le temps de transmission d'un message nécessite une unité de temps

Dans ce cas

Complexité en temps = longueur de la plus longue chaîne de messages

Vérification d'une application répartie

<u>Problème</u> : l'application est non déterministe

- ➔ Multiplication des traces d'exécution
  - Pas de construction exhaustive des états
- ➔ Difficulté (impossibilité) de reproduire une exécution
  - Extrêmement difficile à débugger

Nécessité de prouver l'application *a priori* sans construction de tous les états

- ➔ Utilisation de méthodes (inductives) basées sur des propriétés
  - Invariants et propriétés stables pour la sûreté
  - Fonctions de progrès pour la vivacité

Propriétés de sûreté et de vivacité

Sûreté :

rien de mauvais ne se produira dans l'exécution de l'application

- ➔ *Exclusion Mutuelle* : toujours au plus un processus en section critique
- ➔ *Détection de la Terminaison* : si on détecte la terminaison de l'application, alors celle-ci est réellement terminée (pas de fausse détection)

Vivacité :

quelque chose de bien finira par se produire dans l'exécution

- ➔ *Exclusion Mutuelle* : si un processus demande la section critique, il finira par l'obtenir
- ➔ *Détection de la Terminaison* : si l'application se termine, alors cette terminaison sera détectée