

# ***Modélisation conceptuelle des Systèmes Distribués***

Eric Cariou

Master Technologies de l'Internet 1<sup>ère</sup> année

*Université de Pau et des Pays de l'Adour  
Département Informatique*

Eric.Cariou@univ-pau.fr

# *Systemes distribués*

- ◆ Système distribué en opposition à système centralisé
- ◆ Système centralisé : tout est localisé sur la même machine et accessible par le programme
  - ◆ Système logiciel s'exécutant sur une seule machine
  - ◆ Accédant localement aux ressources nécessaires (données, code, périphériques, mémoire ...)
- ◆ Système distribué : une définition parmi d'autres
  - ◆ Ensemble d'ordinateurs indépendants connectés en réseau et communiquant via ce réseau
  - ◆ Cet ensemble apparaît du point de vue de l'utilisateur comme une unique entité

# *Systemes distribués*

- ◆ Vision matérielle d'un système distribué : architecture matérielle
  - ◆ Machine multi-processeurs avec mémoire partagée
  - ◆ Cluster d'ordinateurs dédiés au calcul/traitement massif parallèle
  - ◆ Ordinateurs standards connectés en réseau
- ◆ Vision logicielle d'un système distribué
  - ◆ Système logiciel composé de plusieurs entités s'exécutant indépendamment et en parallèle sur un ensemble d'ordinateurs connectés en réseau
- ◆ Dans ce cours
  - ◆ Conception logicielle des systèmes distribués
  - ◆ Par défaut sur une architecture matérielle de type ordinateurs connectés en réseau

# *Introduction*

- ◆ De par leur nature, les systèmes distribués sont dans un cadre différent par rapport aux systèmes centralisés
- ◆ Concurrence
  - ◆ Les éléments formant le système s'exécutent en parallèle et de manière autonome
  - ◆ Pas d'état ou d'horloge globale commune
- ◆ Points de problèmes de fiabilité en nombre accru
  - ◆ Problème matériel d'une machine
  - ◆ Problème de communication via le réseau
  - ◆ Problème logiciel sur un des éléments du système
- ◆ Communication est un point crucial
  - ◆ Potentiellement non fiable
  - ◆ Temps de communication non négligeables

# *Transparences*

## ◆ Transparence

- ◆ Fait pour une fonctionnalité, un élément d'être invisible ou caché à l'utilisateur ou un autre élément formant le système distribué
  - ◆ Devrait plutôt parler d'opacité dans certains cas ...
- ◆ But est de cacher l'architecture, le fonctionnement de l'application ou du système distribué pour apparaître à l'utilisateur comme une application unique cohérente
- ◆ L'ISO définit plusieurs transparences (norme RM-ODP)
  - ◆ Accès, localisation, concurrence, réplication, mobilité, panne, performance, échelle

# *Transparences*

- ◆ **Transparence d'accès**
  - ◆ Accès à des ressources distantes aussi facilement que localement
  - ◆ Accès aux données indépendamment de leur format de représentation
- ◆ **Transparence de localisation**
  - ◆ Accès aux éléments/ressources indépendamment de leur localisation
- ◆ **Transparence de concurrence**
  - ◆ Exécution possible de plusieurs processus en parallèle avec utilisation de ressources partagées
- ◆ **Transparence de réplication**
  - ◆ Possibilité de dupliquer certains éléments/ressources pour augmenter la fiabilité

# *Transparences*

- ◆ Transparence de mobilité
  - ◆ Possibilité de déplacer des éléments/ressources
- ◆ Transparence de panne
  - ◆ Doit supporter qu'un ou plusieurs éléments tombe en panne
- ◆ Transparence de performance
  - ◆ Possibilité de reconfigurer le système pour en augmenter les performances
- ◆ Transparence d'échelle
  - ◆ Doit supporter l'augmentation de la taille du système (nombre d'éléments, de ressources ...)

# *Transparences*

- ◆ Un système donné va offrir un certain nombre de transparences
  - ◆ Souvent au minimum transparences de localisation, d'accès et de concurrence
- ◆ Système distribué ouvert
  - ◆ Peut être étendu en nombre d'éléments matériels le constituant
  - ◆ Possibilité d'ajouts de nouveaux services ou de ré-implémentation de services existants au niveau logiciel
  - ◆ Fonctionnement se base sur des interfaces d'interactions clairement définies

# *Algorithmique distribuée*

- ◆ Si l'on veut développer des logiciels fiables et robustes dans un contexte distribué, il faut
  - ◆ Tenir compte des spécificités des systèmes distribués
    - ◆ Notamment les temps de communication et la multiplication des possibilités de pannes
  - ◆ Développer des algorithmes dédiés aux systèmes distribués : but de *l'algorithmique distribuée*
    - ◆ Exclusion mutuelle distribuée
    - ◆ Transaction distribuée
    - ◆ Diffusion causale
    - ◆ Vote, consensus
    - ◆ ....

# *Algorithmique distribuée*

- ◆ Algorithmique distribuée
  - ◆ Développement d'algorithmes dédiés aux systèmes distribués et prenant en compte les spécificités de ces systèmes
  - ◆ On y retrouve notamment des adaptations de problèmes classiques en parallélisme
    - ◆ Exclusion mutuelle, élection (d'un maître) ...
  - ◆ Mais aussi des problèmes typiques des systèmes distribués
    - ◆ Horloge globale, état global, diffusion causale, consensus ...
  - ◆ S'intéresse principalement à deux grandes familles de pbs
    - ◆ Synchronisation et coordination entre processus distants
    - ◆ Entente sur valeurs communes et cohérence globale dans un contexte non fiable (crash de processus, perte de messages ...)

# *Algorithmique distribuée*

- ◆ Les algorithmes distribués s'appuie sur des caractéristiques du système
  - ◆ Caractéristiques des éléments formant le système
    - ◆ Fiable, pouvant se planter, pouvant envoyer des messages erronés ...
  - ◆ Caractéristiques de la communication
    - ◆ Fiable ou non fiable, temps de propagation borné ou pas...
- ◆ Un algorithme distribué se base donc sur un modèle de système distribué qui caractérise
  - ◆ Les processus
  - ◆ La communication
  - ◆ Les pannes

# *Algorithmique distribuée*

- ◆ Système distribué
  - ◆ Ensemble d'éléments logiciels s'exécutant en parallèle
    - ◆ On parlera de processus pour ces éléments logiciels
- ◆ Différences entre modèles de systèmes en algorithmique parallèle et distribuée
  - ◆ Parallèle : sur une même machine
    - ◆ Les processus ont accès à une mémoire locale commune
    - ◆ Les processus s'exécutent par rapport à la même horloge
  - ◆ Distribué : sur un ensemble de machines
    - ◆ Pas de mémoire partagée commune
    - ◆ Pas d'horloge commune
    - ◆ Temps de propagation des messages entre processus distants est non nul

# *Modèles conceptuels de systèmes distribués*

- ◆ Éléments de base d'un système
  - ◆ Processus communiquant
  - ◆ Canaux de communication
- ◆ On modélise un système distribué selon plusieurs dimensions
  - ◆ Modèle d'interaction
  - ◆ Modèle de fautes
  - ◆ Modèle de sécurité
    - ◆ *Non traité dans ce cours*

# *Processus*

- ◆ **Processus**
  - ◆ Élément logiciel effectuant une tâche, un calcul
    - ◆ Exécution d'un ensemble d'instructions
    - ◆ Une instruction correspond à un événement local au processus
      - ◆ Dont les événements d'émission et de réception de messages
    - ◆ Les instructions sont généralement considérées comme atomiques
  - ◆ Il possède une mémoire locale
  - ◆ Il possède un état local
    - ◆ Ensemble de ses données et des valeurs de ses variables locales
  - ◆ Il possède un identifiant qu'il connaît
  - ◆ Pas ou peu de connaissance des autres processus du système et de leur état
  - ◆ Les processus d'un système s'exécutent en parallèle<sup>14</sup>

# Canaux

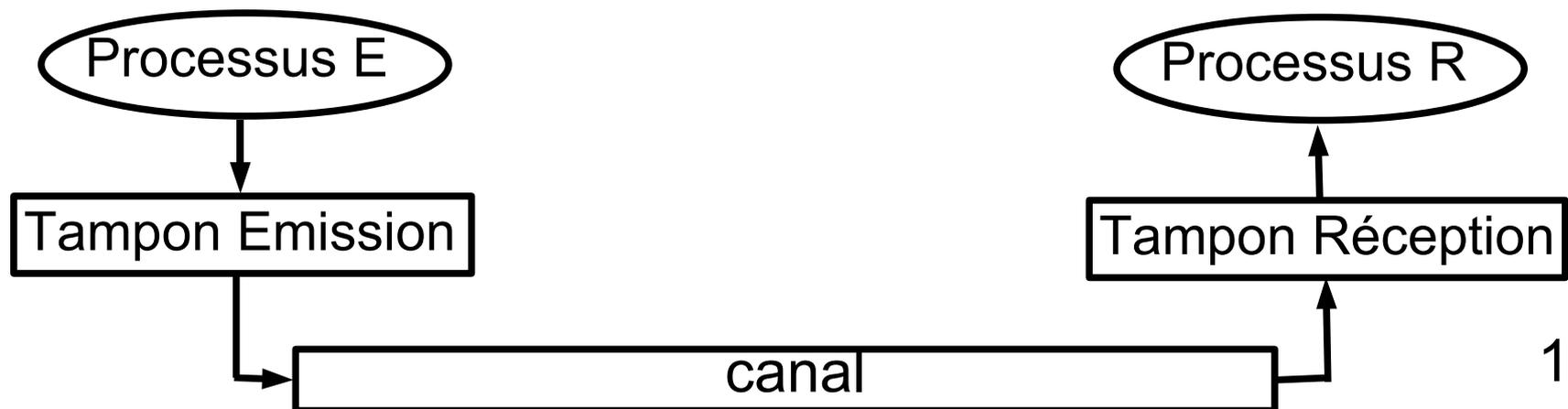
- ◆ Canal de communication
  - ◆ Canal logique de communication point à point
    - ◆ Pour communication entre 2 processus
    - ◆ Transit de messages sur un canal
  - ◆ Caractéristiques d'un canal
    - ◆ Uni ou bi-directionnel
    - ◆ Fiable ou non : perd/modifie ou pas des messages
    - ◆ Ordre de réception par rapport à l'émission
      - ◆ Ex. : FIFO = les messages sont reçus dans l'ordre où ils sont émis
    - ◆ Synchrones ou asynchrones
      - ◆ Synchrones : l'émetteur et le récepteur se synchronisent pour réaliser l'émission et/ou la réception
      - ◆ Asynchrones : pas de synchronisation entre émetteur et récepteur
    - ◆ Taille des tampons de message cotés émetteur et récepteur
      - ◆ Limitée ou illimitée

# Canaux

- ◆ Caractéristique d'un canal
  - ◆ Modèle généralement utilisé
    - ◆ Fiable, FIFO, tampon de taille illimitée, asynchrone (en émission et réception) et bidirectionnel
    - ◆ Variante courante avec réception synchrone
  - ◆ Exemple : modèle des sockets TCP
    - ◆ Fiable
    - ◆ FIFO
    - ◆ Bidirectionnel
    - ◆ Synchrone pour la réception
      - ◆ On reçoit quand l'émetteur émet
      - ◆ Sauf si données non lues dans le tampon coté récepteur
    - ◆ Asynchrone en émission
      - ◆ Emetteur n'est pas bloqué quand il émet quoique fasse le récepteur

# Canaux

- ◆ Performances d'un canal
  - ◆ Latence
    - ◆ Délai entre l'émission d'un message et sa réception
  - ◆ Bande passante du réseau
    - ◆ Nombre d'informations transitant par unité de temps
  - ◆ La gigue
    - ◆ Variation de temps pour délivrer des messages d'une série
      - ◆ Important dans le cadre des données multi-média nécessitant des synchronisations
- ◆ Liaison canal/processus (canal uni-directionnel)



# *Modèles d'interaction*

- ◆ Modèle d'interaction d'un système distribué
  - ◆ Contient un modèle de canal
  - ◆ Contient un modèle temporel : deux modèles principaux
    - ◆ Système distribué synchrone : possède les 3 caractéristiques suivantes
      - ◆ Le temps pour exécuter une étape du processus est compris entre une borne min et une borne max connues
      - ◆ Un message émis est reçu dans un délai max connu
      - ◆ L'horloge locale d'un processus dérive au plus d'une certaine durée connue et bornée du temps réel
    - ◆ Système distribué asynchrone : il n'y aucune borne
      - ◆ Sur la durée d'exécution d'une étape du processus
      - ◆ Sur le délai de réception d'un message
      - ◆ Sur la dérive de l'horloge locale d'un processus par rapport au temps réel

# *Synchrone / Asynchrone*

- ◆ Un modèle synchrone est un modèle où les contraintes temporelles sont bornées
  - ◆ On sait qu'un processus évoluera dans un temps borné
  - ◆ On sait qu'un message arrivera en un certain délai
  - ◆ On connaît la limite de dérive des horloges locales
- ◆ Un modèle asynchrone n'offre aucune borne temporelle
  - ◆ Modèle bien plus contraignant et rendant impossible ou difficile la mise en oeuvre de certains algorithmes distribués

# *Modèle de fautes*

- ◆ Modèle de fautes : 3 grands types de fautes ou pannes
  - ◆ Franches : le processus ne fait plus rien
  - ◆ Par omission : des messages sont perdus ou non délivrés
  - ◆ Arbitraires ou byzantines : le processus renvoie des valeurs fausses et/ou fait « n'importe quoi »
    - ◆ Cas de fautes les plus complexes à gérer
    - ◆ Les autres fautes peuvent être considérées comme des cas particuliers des fautes byzantines
- ◆ Processus correct
  - ◆ Processus non planté, qui ne fait pas de fautes
  - ◆ Dans le cas où on considère les reprises après erreurs et la relance de processus : processus qui pouvait être incorrect précédemment mais qui est correct maintenant

# *Fautes par omission*

- ◆ Fautes par omission
  - ◆ Processus ou canal ne réalise pas une action
  - ◆ Classification des fautes
    - ◆ Fail-stop : un processus s'arrête et reste hors-service. Les autres processus peuvent détecter que ce processus est planté.
    - ◆ Crash : un processus s'arrête et reste hors-service mais les autres processus ne peuvent pas détecter que le processus est planté.
    - ◆ Omission du canal : un message positionné dans un tampon de sortie d'un processus n'arrive jamais dans le tampon d'entrée du destinataire
    - ◆ Omission en émission : un processus envoie un message mais il n'est pas placé dans le tampon d'émission
    - ◆ Omission en réception : un message est placé dans le tampon d'entrée d'un processus mais le processus ne le reçoit pas 21

# *Fautes arbitraires / byzantines*

- ◆ Fautes arbitraires
  - ◆ Appelées aussi fautes byzantines
  - ◆ Fautes quelconques et souvent difficilement détectables
  - ◆ Processus
    - ◆ Calcul erroné ou non fait
    - ◆ État interne faux
    - ◆ Valeur fausse renvoyée pour une requête
  - ◆ Canal
    - ◆ Message modifié, dupliqué voire supprimé
    - ◆ Message « créé »
    - ◆ En pratique peu de fautes arbitraires sur les canaux car les couches réseaux assurent la fiabilité de la communication
- ◆ Fautes arbitraires : classement en 2 catégories
  - ◆ Malicieuses : erreurs volontaires (virus, attaques ...)
  - ◆ Naturelles : problème non voulu, non déclenché

# *Fautes temporelles*

- ◆ Fautes temporelles
  - ◆ Uniquement pour un système distribué synchrone : dépassement des bornes temporelles
    - ◆ Une étape du processus s'exécute en plus de temps que la borne prévue
    - ◆ L'horloge locale dérive d'une valeur supérieure à la borne prévue
    - ◆ Un message émis est reçu après un délai supérieur à la borne prévue

# Détecteurs de fautes

- ◆ Détecteur de fautes
  - ◆ Éléments associés aux processus « essayant » de déterminer l'état des autres processus
    - ◆ Soupçon de fautes de la part d'un processus
    - ◆ Généralement, on se base sur une communication fiable
      - ◆ Messages non perdus
      - ◆ Pouvant être asynchrone : temps de propagation quelconque
- ◆ Classification des caractéristiques des détecteurs
  - ◆ Complétude (*completeness*) : un processus fautif est soupçonné de l'être par un autre processus
  - ◆ Exactitude (*accuracy*) : un processus correct n'est pas soupçonné d'être en panne
  - ◆ Deux variantes à chaque fois
    - ◆ Forte : par tous les processus
    - ◆ Faible : par au moins un processus

# *Classification détecteurs de fautes*

- ◆ Complétude
  - ◆ Forte : tout processus fautif finit par être soupçonné par tous les processus corrects
  - ◆ Faible : tout processus fautif finit par être soupçonné par un processus correct
- ◆ Les deux complétudes sont équivalentes
  - ◆ La faible implique la forte
    - ◆ Localement, un détecteur gère la liste des processus qu'il soupçonne comme fautifs
    - ◆ Les détecteurs s'échangent leurs listes via diffusion
      - ◆ Au bout d'un certain temps, un processus soupçonné fautif sera référencé comme fautif par tous les processus
    - ◆ Hypothèse valable si diffusion fiable
  - ◆ Par principe, la forte implique la faible

# Classification détecteurs de fautes

- ◆ Exactitude
  - ◆ Forte : aucun processus correct n'est jamais soupçonné par aucun processus correct
  - ◆ Faible : au moins un processus correct n'est jamais soupçonné par aucun processus correct
- ◆ Exactitude, versions plus faibles : vérifiée au bout d'un certain temps
  - ◆ Finalement forte : au bout d'un certain temps, un processus correct n'est plus jamais soupçonné par aucun processus correct
  - ◆ Finalement faible : au bout d'un certain temps, au moins un processus correct n'est plus jamais soupçonné par aucun processus correct
  - ◆ Finalement : traduction de *eventually*
    - ◆ Pourrait aussi utiliser : ultimement, inévitablement ...
- ◆ Ces versions plus faibles sont vérifiées si les versions « totales » le sont

# Classification détecteurs de fautes

- ◆ Au final, 8 combinaisons

		<i>Exactitude</i>			
		<b>Forte</b>	<b>Faible</b>	<b>Finalement Forte</b>	<b>Finalement Faible</b>
<i>Complétude</i>	<b>Forte</b>	P	S	◇P	◇S
	<b>Faible</b>	Q	W	◇Q	◇W

- ◆ Mais complétude forte équivalent à la faible
- ◆ Reste donc 4 combinaisons, avec
  - ◆ P qui implique S
  - ◆ ◇P qui implique ◇S
  - ◆ P qui implique ◇P
  - ◆ S qui implique ◇S
- ◆ Détecteur parfait (fiable) : classe P
  - ◆ Complétude et exactitude fortes

# Réalisation de détecteurs de fautes

- ◆ Deux grands modes
  - ◆ Actif : *ping*
    - ◆ Un processus envoie périodiquement un message à un autre processus et attend une réponse de ce dernier
    - ◆ Si réponse pas reçue avant un certain délai de garde, on considère le processus mort
  - ◆ Passif : *heartbeat*
    - ◆ Un processus diffuse périodiquement un message à tous les autres processus
    - ◆ Si au bout d'un délai de garde, on a pas reçu un message d'un processus, on le considère comme mort
- ◆ En pratique, on combine souvent les deux
  - ◆ Heartbreak en fonctionnement nominal
  - ◆ Ping avec les processus qu'on soupçonne d'être morts

# *Réalisation de détecteurs de fautes*

- ◆ Ex. pour un contexte de communication fiable
  - ◆ Pour ne pas avoir problème de ne pas pouvoir différencier une perte de message d'un message non envoyé
- ◆ Système synchrone
  - ◆ Peut construire un détecteur de fautes fiable (parfait)
    - ◆ Grace au délai borné de réception des messages, on sait qu'on recevra un message dans un certain délai si le processus est vivant
- ◆ Système asynchrone
  - ◆ Temps de propagation des messages non bornés
    - ◆ Difficulté (voire impossibilité) de différencier un message lent d'un processus mort
  - ◆ En pratique : choix important du délai de garde
    - ◆ Trop petit : soupçonne avec erreur des processus
    - ◆ Trop grand : temps long avant de détecter la mort d'un processus
  - ◆ Détecteur parfait est non réalisable dans un système asynchrone
    - ◆ Mais souvent on peut se contenter de détecteurs imparfaits