

# **Master 1 Informatique**

## **Algorithmes et Systèmes Distribués**

Erwan Fabiani (UBO)  
[fabiani@univ-brest.fr](mailto:fabiani@univ-brest.fr)

# Description de l'UE

---

- Volume horaire : 44h

- 6 séances de CM
- 8 séances de TD
- 8 séances de TP

- Évaluation

- Un écrit de 2 h (2/3), tous documents autorisés
- Un examen pratique de 30 minutes (1/3), sur PC fixe, tous documents écrits autorisés, accès aux fichiers personnels autorisé

- Références principales

- *Distributed Algorithms*, Nancy Lynch, Morgan Kaufman
- Une partie des transparents proviennent du cours de Bernard Pottier

# Contenu des CM

---

- Introduction, contexte, modèles, problématique
- Le langage GO
- Algorithmes distribués dans le modèle synchrone
- Algorithmes distribués dans le modèle asynchrone
- Exemples applicatifs dans le domaine des réseaux
- Algorithme pour la gestion du temps et l'exclusion mutuelle distribuée

# Une définition générale

---

- Systèmes distribués :
  - Systèmes composés d'un ensemble de processus autonomes coopérant pour parvenir à la réalisation d'un objectif
  - Connectés par un réseau de communication
    - dans une zone géographique importante
    - sur un réseau local,
    - sur une puce
  - Utilisant un langage de communication compatible
    - Protocoles
  - Interactions spécifiées via des échanges de messages explicites plutôt que par des partages d'espace mémoire
  - Les processus sont d'importance égale
    - pas de centralisation fixe
    - mais attribution de rôle possible
  - Un système distribué gère les interactions entre les processus

# De l'architecture à l'algorithme

---

- Architecture distribuée ?
  - Un ensemble de nœuds de calcul matériels
  - Un dispositif de communication matériel entre ces nœuds de calcul (NOC, câble, radio, ...)
- Système distribué ?
  - Un ensemble de processus
  - Un ensemble de services assurant la communication entre ces processus au-dessus d'une architecture distribuée
- Algorithme distribué ?
  - Un algorithme divisé en processus communicants
  - Exécuté sur une architecture distribuée
  - Utilisant les services offerts par le système distribué sous-jacent

# Exemples de nœuds de calcul et d'interactions

---

- Cœurs de processeur sur un multicœur interagissant via un NOC (Network On Chip)
- Réseaux de capteurs
- Processeurs sur une carte multiprocesseurs, communications par bus
- Machines dans un réseau local, communication par ethernet
- Routeurs, set-top box, serveurs, clients dans un réseau internet

# Objectifs d'un système distribué

---

- La performance : rapidité du temps de traitement via
  - la répartition des tâches
  - la parcimonie dans les messages échangés
- L'extensibilité (~~scalabilité~~) :
  - Au niveau fonctionnel : ajouter un processus en plus ne nécessite pas de reprogrammer l'ensemble des processus existants
  - Au niveau des performances : ajouter un processus en plus ne dégrade pas le temps de traitement
- La fiabilité :
  - la défaillance d'un processus ou d'un canal de communication n'entraîne pas la défaillance du système
  - mais bien sûr une dégradation des performances est acceptable
  - Ce qu'il faut éviter (Citation de L. Lamport) : *"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable."*

# Systèmes Distribués vs Systèmes concurrents (1)

---

- Dans un système concurrent, les processus sont en concurrence (compétition) pour obtenir une ressource (historiquement, l'unité de traitement)
- Application = des processus + des données
- Données en mémoire centrale
- Processus en cours ou en attente
- Entrelacement de l'exécution des processus



## **Systèmes Distribués vs Systèmes concurrents (2)**

---

- Dans un système distribué :
  - Les processus ne sont pas obligatoirement en concurrence pour l'accès à une ressource
  - Chaque processus dispose d'un espace mémoire privé
  - Les processus échangent des messages
  - Un système distribué est extensible

# Systemes Distribués vs Systemes Parallèles

---

- Systemes Parallèles :
  - Ressources matérielles multiples pour l'amélioration du temps de calcul d'une application
  - Objectifs principaux :
    - la performance,
    - l'efficacité (performances/coût)
- Une application parallélisée n'a pas obligatoirement une nature « parallèle » à l'origine
  - par exemple un programme séquentiel de multiplication de matrice qui est parallélisé
- L'objectif principal d'un système distribué n'est pas obligatoirement la performance, l'application peut avoir une nature intrinsèquement distribuée
  - par exemple le routage dans un réseau

# Distribué, concurrent, parallèle : points communs

---

- Description de comportement :
  - la simultanéité d'exécution de processus
  - les communications entre processus
  - => Existence de modèles, algorithmes, langages permettant de décrire ces trois types de modèles
- Avantages par rapport à une conception séquentielle
  - Gain conceptuel
  - Performances
  - Tolérance aux défaillances
- Problèmes potentiels
  - Indéterminisme
  - Instabilité

# Problèmes à résoudre dans un système distribué (1)

---

- Distribution des tâches aux processus
  - Choix de rôles particuliers pour certains processus
  - Adaptation automatique en cas de défaillance d'un processus choisi
- Répartition mémoire dans les processus
  - Acheminement des informations
    - Vers un seul destinataire (monocast)
    - Vers un groupe (multi-cast)
    - En diffusion à tous (broadcast)
  - Compromis entre
    - Capacité de mémorisation importante (mais actualisation rare)
    - Echange de message fréquent (mais coût de communication important)
- Résolution des accès concurrents
  - Pour des ressources partagées (périphériques, accès réseau externe, ...)

# Problèmes à résoudre dans un système distribué (2)

---

- Fiabilisation des communications
  - Pertes de messages
  - Délais d'acheminements
    - Variables
    - Non prévisibles
    - Non bornables
  - Interruptions
    - Défaillance d'un canal de communication
    - Définitive ou temporaire
  - Ordre aléatoire d'arrivée des messages

## Problèmes à résoudre dans un système distribué (3)

---

- Eviter les interblocages (équilibre non souhaité)
  - Deadlock (inactif) : les processus sont bloqués en attente d'un évènement (message, action) qui n'arrivera jamais, puisque tous les processus sont bloqués
  - Livelock (actif) : les processus sont actifs, mais retraits dans un ensemble d'états dont il ne peuvent sortir, leurs actions combinées ne permettant pas de faire évoluer le système

## Problèmes à résoudre dans un système distribué (3)

---

- Eviter les interblocages (équilibre non souhaité)
  - Interblocages accentués par une programmation uniforme
    - Exemple : 2 drones en face à face dans un couloir aérien, avec seul changement d'altitude en vol stationnaire
    - Programme d'évitement de collision avec obstacle mobile :
      - Arrêt, attente que l'altitude soit dégagée pour avancer => Deadlock
      - Arrêt, montée en altitude de 50 m, test obstacle
        - Si obstacle et altitude max atteinte, descente de 50 m, puis test obstacle
        - Si obstacle et altitude max non atteinte, montée de 50 m, puis test obstacle
      - => Livelock
  - Résolution par une prise de décision différenciée (aléatoire)

## Problèmes à résoudre dans un système distribué (4)

---

- Synchronisation
  - Faire en sorte que les processus soient dans un même état d'avancement de l'algorithme
- Terminaison (équilibre souhaité)
  - Détecter que l'objectif est atteint
  - Faire en sorte que tous les processus soient dans un état stable à la terminaison
  - Si les traitements de l'algorithmes ont tous été effectués mais que des processus ne le savent pas, l'algorithme n'est pas correct



# Problèmes à résoudre dans un système distribué (5)

---

## □ Observation du système

- Il n'est pas possible de connaître l'état global d'un système à un instant particulier
  - Cela implique une centralisation et des communications
  - Cela implique que
    - L'état centralisé observé n'est pas l'état instantané
    - L'état centralisé observé ne correspond pas forcément à un état global ayant existé
- La mise en place de processus d'observation peut modifier le comportement du système :
  - synchronisation par effet de bord
- Un système peut donc fonctionner quand on l'observe, et ne plus fonctionner quand on ne l'observe plus
  - Comment détecter si le système est dans un état conforme ?
  - Comment observer sans modifier le comportement ?

# Difficultés pour résoudre les problèmes

---

- Topologies et nombre de processus variables
  - Topologie : schéma d'organisation des canaux de communication entre processus
  - Variabilité en cas de mobilité des processus
- Données d'entrées indépendantes et distribuées
  - Observation et récupération locale des données
  - Redondances des données observées
- Décalages temporels des programmes
  - Vitesses relatives différentes
  - Matériels hétérogènes

# Difficultés pour résoudre les problèmes

---

- Définir le types des défaillance supportées
  - Pertes de message,
  - Arrêt d'une machine,
  - Comportements aléatoires.
- Non Déterminisme
  - Le contexte influe sur le fonctionnement du système
  - Non reproductibilité des erreurs (bugs)
- Notion de temps ?
  - Impossibilité de percevoir un temps commun (ordre total)
  - Possibilité de percevoir un ordre partiel

# Problématique de l'ordre (1)

---

- Ordre total et ordre partiel
  - Ordre total : dans un programme séquentiel, trace des instructions exécutées dans l'ordre (« artificiel »)
  - Dans un système distribué :
    - Ordre local à chaque processus
    - Synchronisation entre processus
    - => ordre partiel

# Problématique de l'ordre (2)

---

- Ordre partiel :
  - Suite d'ensembles d'états dans lesquels l'ordre d'exécution entre processus est non déterministe
  - Séparés par des points de synchronisation qui garantissent un ordre partiel pour les ensembles d'état successifs
  - L'ordre observé peut-être différent d'une exécution à l'autre.
  - Il doit seulement ne pas être en contradiction avec l'ordre partiel initial : compatible avec la synchronisation ou les messages échangés

# Problématique de l'ordre (2)

---

## □ Précédence directe

- Un événement  $e1$  précède un événement  $e2$  si
  - $e1$  et  $e2$  ont lieu sur le même processus, et  $e1$  est terminé avant le début de  $e2$
  - Ou  $e1$  correspond à l'émission d'un message, et  $e2$  à la réception de ce même message

## □ Précédence causale

- Un événement  $e1$  précède causalement un événement  $e2$  si il existe une suite de relations de précédence directe entre  $e1$  et  $e2$

# Problématique de l'ordre (3)

---

- Respect de l'ordre causal
  - Ordre causal direct
    - Soient  $m1$  et  $m2$  deux messages avec le même processus émetteur et le même processus destinataire
    - Si  $m1$  est émis avant  $m2$ ,  $m1$  doit arriver avant  $m2$
  - Ordre causal indirect
    - Réception par un processus de messages venant de plusieurs processus émetteurs, les messages étant liés par un ordre causal

# Complexité de conception d'un algorithme distribué

---

- Qu'est-ce qu'un algorithme distribué « conforme » ?
  - Un algorithme qui produit des résultats conformes aux spécification
  - Mais aussi :
    - Un algorithme qui se termine
    - Un algorithme qui ne se bloque jamais sur un interblocage
  - Dans un contexte où :
    - Les bugs ne sont pas forcément reproductibles
    - Le détail des problèmes n'est pas forcément observable
- Comment s'assurer qu'un algorithme distribué est « conforme » ?
  - Preuve de programme : algèbre de processus
    - Prouver par raisonnement que l'algorithme est conforme dans tous les cas
  - Conformité par construction
    - Suivre un modèle de conception qui garantit la terminaison et l'absence d'interblocage
    - Le travail du programmeur se concentre alors sur la conformité des résultats
    - => Nécessité de disposer de modèles de conception



# Modèles de conception

---

- ❑ Concevoir un algorithme distribué est un travail complexe si on ne délimite pas précisément un modèle de conception
- ❑ Le programmeur doit
  - ❑ Découper son application en processus autonomes
  - ❑ Définir les communications entre processus (canaux de communication)
  - ❑ simuler son comportement pour détecter les anomalies
- ❑ Modèle de conception
  - ❑ Comment découper une application en processus ?
  - ❑ Quel est le support abstrait des communications entre processus ?
  - ❑ Y-a-t'il un cadre contraint pour la façon de programmer l'application ?

# Approche méthodologique et pédagogique

---

- Méthode de conception généraliste :
  - Pas d'hypothèses contraignantes sur la nature matérielle du support d'exécution
  - Mise en œuvre d'algorithmes de différente nature : distribués, mais aussi parallèle et concurrents
  
- Objectif de cette UE : savoir écrire un algorithme distribué conforme en se basant sur un modèle de conception

# Expression des communications

---

- Dans un modèle à mémoire commune : communication par variables partagées
- Dans un système distribué : opérateurs explicites de communication
- La nécessité des communications transparait dans le graphe des processus
  - Des arcs symbolisent
    - une relation d'ordre (terminaison)
    - des communications
  - La signification du graphe dépend du contexte

# Modélisation des communications

---

- Modes de connexion des communications
  - Mémoire partagée
  - Communication point à point
  - Diffusions (broadcast, multicast)
- Modèle temporel des communications
  - Système synchrone (physiquement)
  - Synchronisme induit (par programmation)
  - Asynchronisme complet
- Modèle de défaillances des communications
  - Perte de message
  - Ordre aléatoire
  - Coupure de communication

# Modèles de communication par messages

---

- Mode message synchrone
  - Canal bloquant de contenance nulle
  
- Mode message asynchrone
  - Producteurs-consommateurs
  - Tampons de stockage des messages
  
- Mode invocation à distance (« rendez vous » étendu)
  - Appel de procédure d'un processus avec passage de paramètres
  - Attente des résultats de la procédure

# Communication par messages synchrone

---

- ❑ L'émetteur et le récepteur sont en même temps dans l'état de communication (d'où le terme synchrone)
- ❑ L'émetteur émet son message et attend la fin de réception par le récepteur
- ❑ Le récepteur qui attend un message est bloqué jusqu'à son arrivée
- ❑ Avantages :
  - ❑ émetteur et récepteur sont dans un état connu de leur évolution
  - ❑ la communication sert aussi à la synchronisation
  - ❑ acquittement implicite : si le processus est débloqué, c'est que le message est bien passé
- ❑ Inconvénient :
  - ❑ fort couplage temporel entre les correspondants
- ❑ Rendez-vous simple (car on ne transmet qu'un seul message pendant le rendez-vous)

# Communication par messages asynchrone

---

- L'émetteur et le récepteur traitent un message à des instants différents (d'où le terme asynchrone)
- L'émetteur émet son message et n'attend pas la fin de réception par le récepteur
- Le récepteur a un rythme autonome de réception, avec deux modes
  - Réception bloquante s'il n'y a pas de message
  - Réception non bloquante, avec un témoin de réception
- Schéma producteur-consommateur avec un tampon à l'émission et un tampon de réception
- Avantage : indépendance temporelle des correspondants
- Inconvénients :
  - pas d'acquiescement implicite
  - pas de relation entre les états de l'émetteur et du récepteur
  - difficultés pour le traitement et la correction des erreurs

# Communication par invocation à distance (rendez-vous étendu)

---

- Un processus émetteur demande l'exécution d'une procédure qui est un composant du processus récepteur
  - premier message : passage des paramètres par l'émetteur
  - second message : passage des résultats par le récepteur
- L'émetteur attend la fin de l'exécution avant de poursuivre
  - Il peut ne pas y avoir de résultat
- Avantages :
  - sémantique précise, celle de l'appel de procédure
  - les correspondants sont dans des états connus
- Inconvénient :
  - fort couplage temporel entre les deux processus pendant l'exécution de l'appel de procédure
- Différent de l'appel de procédure à distance qui est à un niveau physique (on sait que la procédure appelée à distance est sur un autre site) et non à un niveau logique



# Désignation et appel d'une communication

---

- Désignation directe ou indirecte
  - Directe : nom du processus correspondant
    - *envoyer(message, nom\_de\_processus)*
  - Indirecte : relais intermédiaire
    - nom d'un canal, d'un port, ...
    - *envoyer(message, nom\_de\_canal)*
- Appels symétriques ou asymétriques
  - Symétrique :
    - l'émetteur et le récepteur se nomment directement ou indirectement
    - *envoyer(message, nom\_recepteur)* et *recevoir(message, nom\_émetteur)*
    - *envoyer(message, nom\_de\_canal)* et *recevoir(message, nom\_de\_canal)*
  - Asymétrique :
    - le récepteur ne nomme pas de source : *attendre(message)*
    - Il accepte des messages de tout processus (ou canal)
    - adapté au modèle client -serveur

# Trois types d'interactions entre processus

---

## □ Synchronisation

- Règles et mécanismes pour assurer et contrôler une évolution correcte des processus
- Blocage des processus « en avance »

## □ Coopération

- Chaque processus a besoin de coopérer avec les autres pour la réalisation de sa tâche
- Transmission de valeurs

## □ Compétition

- Chaque processus « ignore » a priori l'existence des autres et entre en compétition pour l'accès aux ressources
- Verrouillage des ressources partagées

# Principaux modèles de synchronisation/coopération

---

- Exemples des problèmes les plus fréquents
- Pour ces problèmes, il existe des solutions centralisées ou distribuées
- Certaines de ces solutions seront étudiées en TD et TP
- Principaux problèmes
  - L'exclusion mutuelle
  - La cohorte
  - Le rendez-vous
  - Les producteurs-consommateurs
  - Les lecteurs-rédacteurs
  - L'élection d'un coordinateur
  - Le consensus

# Exclusion mutuelle

---

## □ Définition :

- Accès cohérents à une ressource partagée, qui ne doit être accédée que par un seul processus à la fois
- Cohérence de suites d'actions concurrentes

## □ Exemples :

- Donnée commune (compte client)
- Contrôle de voie unique (trains, réseau, canal d'écran, ...)
- Attribution d'un nom unique

# Cohorte

---

## □ Définition :

- Coopération d'un groupe de taille maximale donnée
- Exclusion mutuelle avec plus d'une ressource

## □ Exemples :

- Groupe de machines pour un calcul coopératif
- Jetons pour l'utilisation d'un logiciel
- Cluster de communication

# Élection d'un coordinateur (leader)

---

## □ Définition

- Désignation d'un processus leader dans une application distribuée
- Le processus leader doit savoir qu'il est élu leader
- Les processus non-leader doivent savoir qu'ils ne le sont pas
- Il n'est pas obligatoire que les processus non-leader sachent qui est le leader

## □ Exemples

- Choix d'un serveur dans un réseau après une défaillance

# Consensus

---

## □ Définition :

- Les processus doivent voter (oui ou non) pour l'acceptation éventuelle d'une action
- L'action est acceptée si et seulement si tous les processus ont voté oui

## □ Exemples :

- Transaction dans une base de données distribuée
- Terminaison d'un traitement coopératif

# Rendez-vous

---

## □ Définition :

- Synchronisation d'un ensemble de processus
- Les processus sont bloqués tant que tous les processus ne sont pas prêts

## □ Exemples :

- Réinitialisation d'un groupe de machines
- Attente de résultats d'observations



# Producteurs-Consommateurs

---

## □ Définition

- Communication asynchrone entre des producteurs et des consommateurs par le biais d'un tampon avec une taille maximale
- Tout message produit est consommé une seule fois
- Écriture bloquante pour un producteur si il souhaite écrire et que le tampon est plein
- Lecture bloquante pour un consommateur si il souhaite lire et que le tampon est vide

## □ Paramètres

- Nombre de producteurs et consommateurs
- Ordre de prélèvement des messages
- Structuration du tampon

## □ Exemples

- Serveur d'impression
- Tampon mémoire
- Service de messagerie

# Lecteurs-Rédacteurs

---

## □ Définition

- Partage de données entre des processus lecteurs et rédacteurs
- Un seul rédacteur à la fois
- Plusieurs lecteurs en parallèles

## □ Exemples

- Base d'informations temps-réel
- Gestion de caches multiples dans les multiprocesseurs
- Mémoire virtuelle répartie