Systèmes et algorithmes répartis

Principes et concepts

Philippe Quéinnec, Gérard Padiou
queinnec@enseeiht.fr

ENSEEIHT
Département Sciences du Numérique

17 octobre 2024

La présence de ♪♪♪ indique un complément audio : cliquer dessus.

Plan

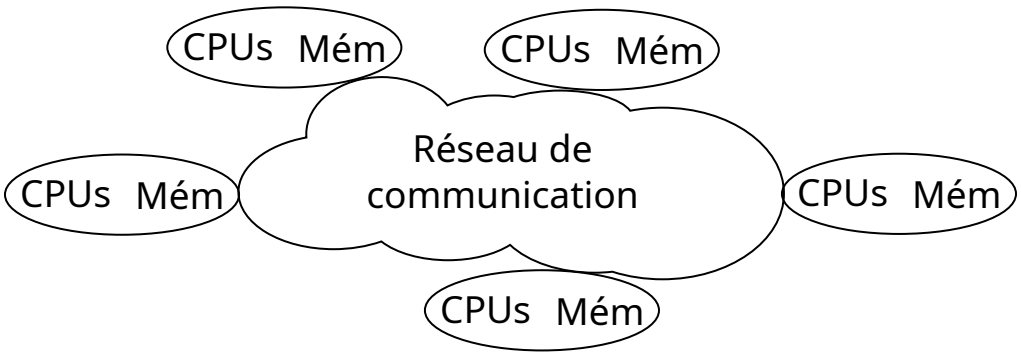
Sources

- G. Padiou, « *Précis de répartition* », 2016, `http://queinnec.perso.enseeiht.fr/Ens/SAR/precis.pdf`
  Référencé par *[Précis 1.5 p.13]* (section, page)
- M. Raynal, « *Distributed Algorithms for Message-Passing Systems* », « *Fault-Tolerant Agreement in Synchronous Message-passing Systems* » et « *Communication and Agreement Abstractions for Fault-Tolerant Asynchronous Distributed Systems* », 2010–2012
- S. Krakowiak, « *Algorithmique et techniques de base des systèmes répartis* », `http://lig-membres.imag.fr/krakowia/Files/Enseignement/M2R-SL/SR/`
- A.D. Kshemkalyani, M. Singhal, « *Distributed Computing : Principles, Algorithms, and Systems* », 2008
  `http://www.cs.uic.edu/~ajayk/DCS-Book`

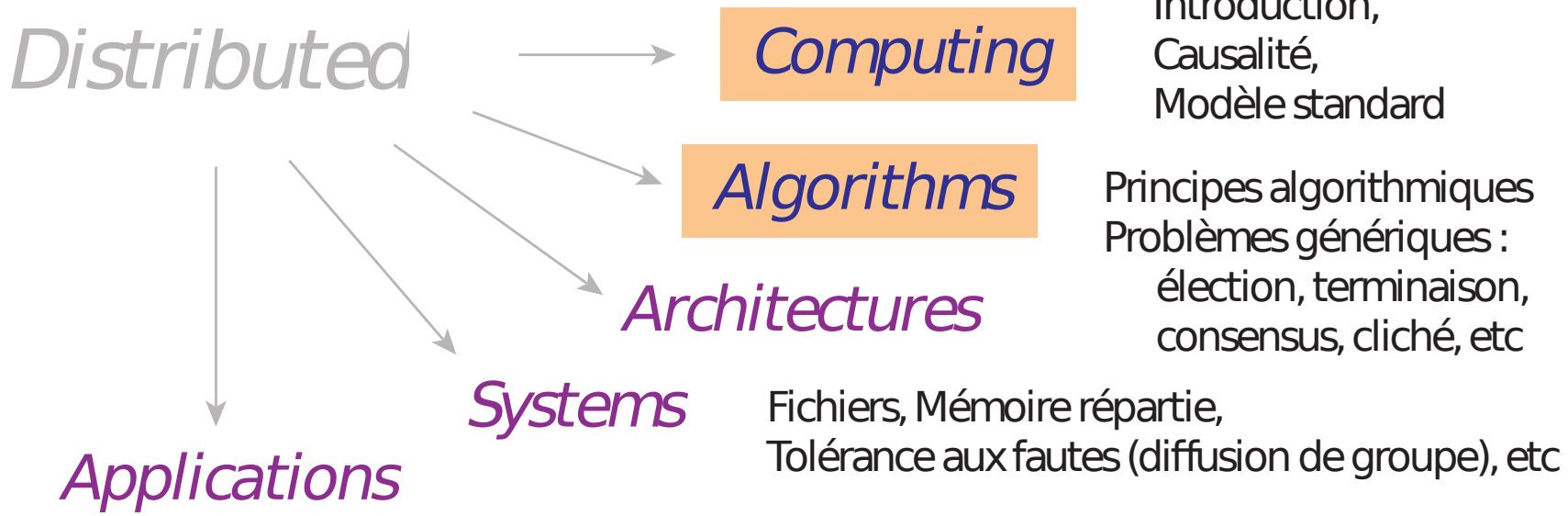
Préambule : tendance

Répartition ≡ communication entre objets informatisés

- Actuellement : ordinateurs + téléphones + tablettes toujours connectés
- L'Internet des objets (*The Internet of things*)
  - 20 à 30 milliards d'appareils connectés en 2020
  - du porte-clefs au réfrigérateur en passant par les plantes
- L'informatique dans les nuages (*cloud computing*) : l'accès pour tous aux ressources/services informatiques



Préambule : de quoi allons nous parler ? ♪♪♪



Plan du cours

I. Principes et concepts
II. Modèle standard et principes algorithmiques
III. Causalité et datation
IV. Problèmes génériques
V. Grande échelle, pair-à-pair
VI. Consensus, détecteur de défaillances
VII. Données réparties
VIII. Construction d'objets concurrents
IX. Tolérance aux fautes
X. Simulation répartie

Cours II à VI : Quiz dans la semaine suivante, +0,2 à 0,3 par quiz à l'examen si assez correct (voir moodle)

Plan

Intérêts

♪♪♪

Apports de la répartition

- Accès aux ressources distantes et partage :
  - ressources physiques : imprimantes, traceurs...
  - ressources logiques : fichiers
  - données : textuelles, audio, images, vidéo
- Répartition géographique
- Puissance de calcul
- Disponibilité
- Flexibilité

[Précis 1.3 pp.9–10]

Modèle centralisé ou réparti

Modèle centralisé

Les processus se partagent des ressources critiques ou pas, et communiquent par mémoire partagée

Modèle réparti

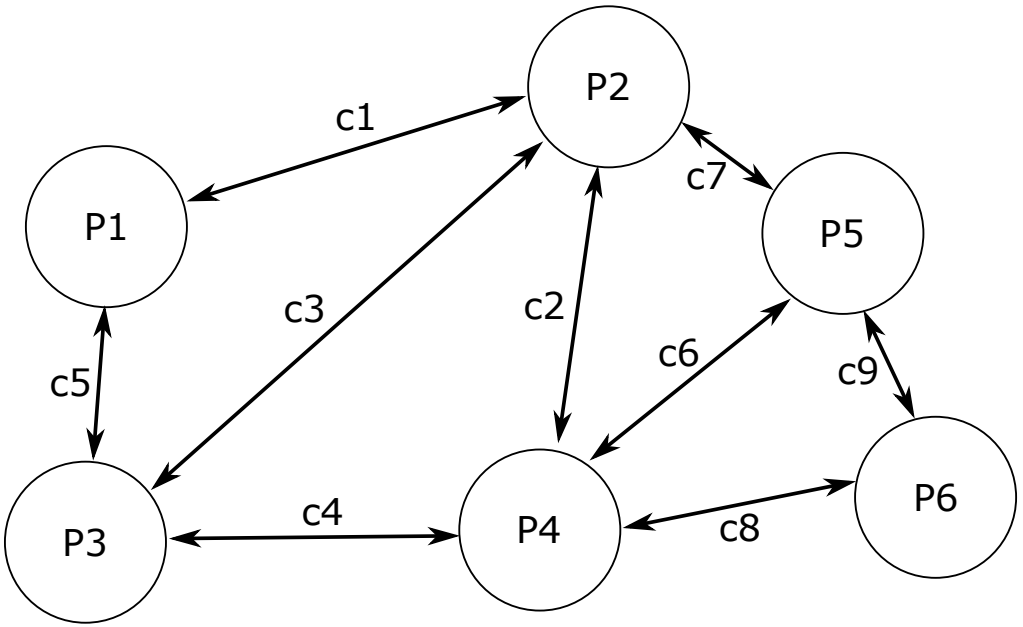
Les processus échangent des données par messages

Exemple : découvrir le graphe de communication ♪♪♪

Le problème

On considère un ensemble de sites connectés deux à deux par des canaux bidirectionnels.

Comment un site peut-il apprendre la structure du graphe ?



Hypothèses

♪♪♪

Connaissance initiale

Chaque site connaît son identité (id_i) et l'identité de ses voisins $voisins_i = \{id_x, id_y, \ldots\}$.
Le couple (id_i, id_j) représente le canal entre id_i et id_j (et symétriquement).
Aucun site ne connaît l'identité de tous les sites, ni leur nombre.

Communication

Un site peut envoyer/recevoir un message uniquement à ses voisins.

Découverte : principe de l'inondation ♪♪♪

Le problème redéfini

Comment chaque site id_i peut-il connaître l'ensemble des canaux (id_j, id_k) existants ?

Principe

- Un site i qui veut connaître le graphe envoie sa position $\langle id_i, voisins_i \rangle$ à ses voisins.
- La première fois qu'un site i reçoit un message, il s'active et envoie sa position $\langle id_i, voisins_i \rangle$ à ses voisins.
- La première fois qu'un site i reçoit un message $\langle id_k, voisins_k \rangle$, il le transmet à ses voisins et met à jour sa connaissance du graphe (pour k). Sinon, il l'ignore.
- Un site conclut quand il a reçu un message de tous les sites dont il a eu connaissance via les voisinages.

Découverte : principe de l'inondation ♪♪♪

Le problème redéfini

Comment chaque site id_i peut-il connaître l'ensemble des canaux (id_j, id_k) existants ?

Principe

- Un site i qui veut connaître le graphe envoie sa position $\langle id_i, voisins_i \rangle$ à ses voisins.
- La première fois qu'un site i reçoit un message, il s'active et envoie sa position $\langle id_i, voisins_i \rangle$ à ses voisins.
- La première fois qu'un site i reçoit un message $\langle id_k, voisins_k \rangle$, il le transmet à ses voisins et met à jour sa connaissance du graphe (pour k). Sinon, il l'ignore.
- Un site conclut quand il a reçu un message de tous les sites dont il a eu connaissance via les voisinages.

Algorithme pour le site i

Variables locales au site i :
id_i : son identité (const)
$voisins_i$: ses voisins (const)
$sites\_known_i$: les sites dont il a reçu un message
$channels\_known_i$: les canaux qu'il connaît

```
on start :
  for each id_j ∈ voisins_i do
      send ⟨id_i, voisins_i⟩ to id_j
  end for
  sites_known_i ← {id_i}
  channels_known_i ←
     { (id_i, id_k) :  id_k ∈ voisins_i }

  on reception ⟨id, voisins⟩ :
    if sites_known_i = ∅ then start(); fi
    if id ∉ sites_known_i then        -- premier message de id
      sites_known_i ← sites_known_i ∪ {id}
      channels_known_i ← channels_known_i ∪ { (id, id_k) :  id_k ∈ voisins }
      for each id_j ∈ voisins_i        -- propage le message
          send ⟨id, voisins⟩ to id_j
      end for
      if ∀ (id_j, id_k) ∈ channels_known_i : {id_j, id_k} ⊆ sites_known_i then
          id_i connaît le graphe. FIN
      endif
    endif
```

Questions pas triviales

- Terminaison : tous les sites finissent-ils par atteindre FIN ?
- Terminaison : un site peut-il savoir que les autres ont terminé ?
- Correction : à la terminaison de i, $channels\_known_i$ = le graphe ?
- Correction : après terminaison de tous, $\forall i, j : channels\_known_i = channels\_known_j$?
- Coût en messages ?
- Complexité en temps ?
- Résistance à un arrêt de site ?

Questions pas triviales

♪♪♪

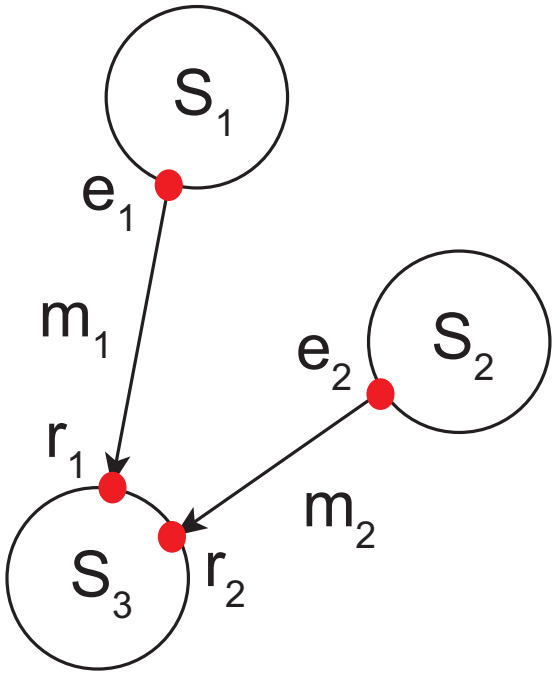
- Terminaison : tous les sites finissent-ils par atteindre FIN ? (oui)
- Terminaison : un site peut-il savoir que les autres ont terminé ? (non)
- Correction : à la terminaison de i, $channels\_known_i$ = le graphe ? (oui)
- Correction : après terminaison de tous, $\forall i, j : channels\_known_i = channels\_known_j$? (oui)
- Coût en messages ? (2 * nombre de sites * nombre de canaux)
- Complexité en temps ? (2 * diamètre)
  (diamètre = $\max_{(i,j)} \min distance(i, j)$)
- Résistance à un arrêt de site ? (on perd la terminaison ; ok si le graphe reste connexe et que tout site fait au moins "start")

Modèle d'exécution plus complexe ♪♪♪

Problèmes...

- $m1$ est-il toujours envoyé avant $m2$ dans toute exécution ?
- $m1$ est-il toujours reçu avant $m2$ dans toute exécution ?
- Peut-on déduire ?

$$date(r_1) < date(r_2)$$
$$\Downarrow ?$$
$$date(e_1) < date(e_2)$$



Fort non-déterminisme : explosion des états possibles

[Précis 1.2 pp.7–9]

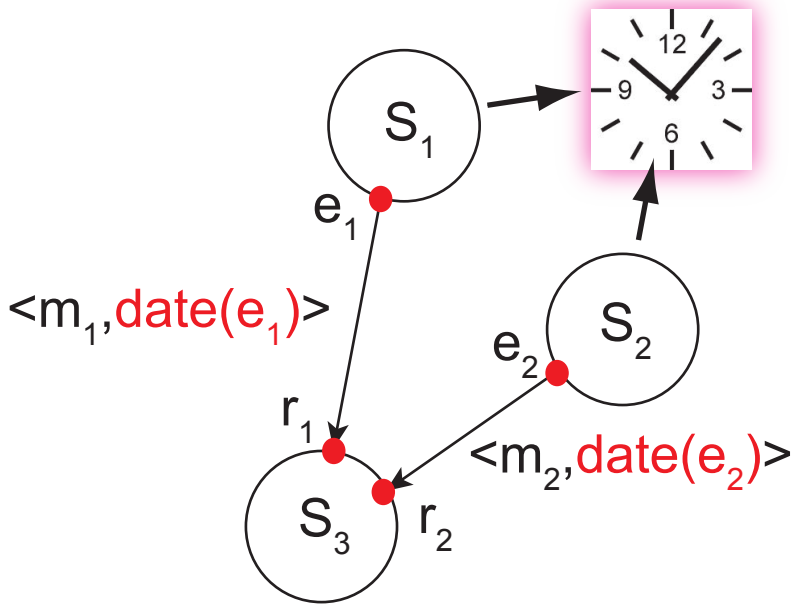
Épine : le temps

♪♪♪

Dates dans messages

$$date(e_1) < date(e_2)$$
$$\Downarrow ?$$
$$e_1 \text{ avant ? } e_2$$

Pas sûr, car
l'horloge n'existe pas ! ! !



Épine : pas de temps global

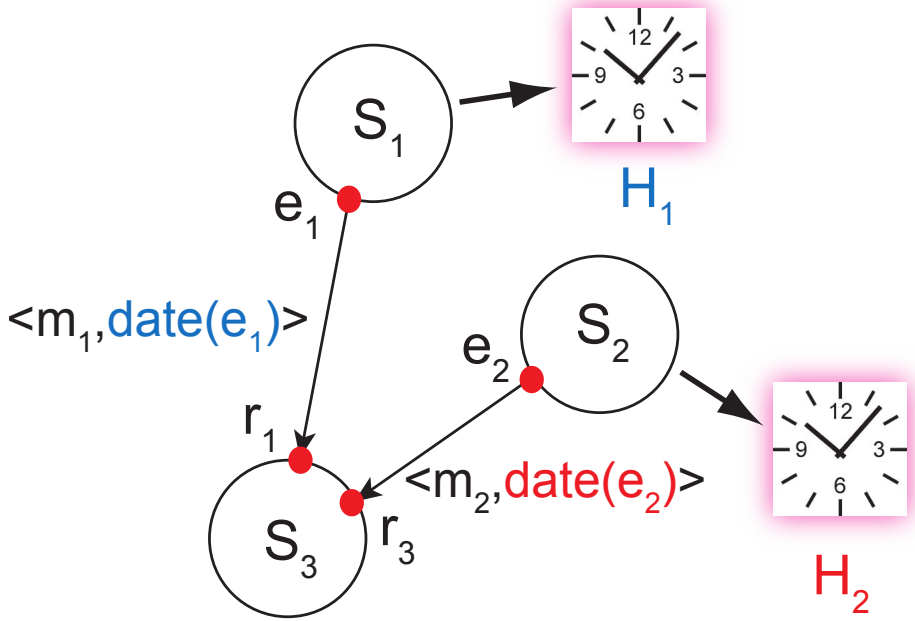
il existe 2 horloges

$$date(e_1) < date(e_2)$$

$$\Downarrow$$

e_1 avant e_2

Si les horloges sont synchronisées !

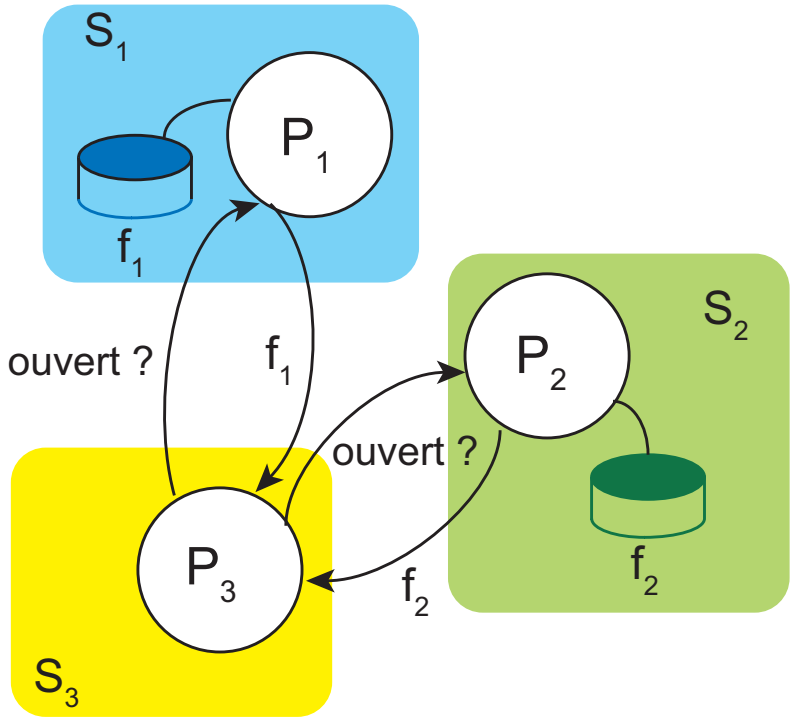


Pas de référentiel temporel unique

Épine : pas d'état global immédiat ♪♪♪

Problème

- P3 veut savoir si P1 ou P2 ont ouvert des fichiers ?
- Connaissance instantanée impossible



Un processus ne peut pas connaître instantanément l'état courant de ses partenaires : pas d'état global immédiat.

Épine : les défaillances

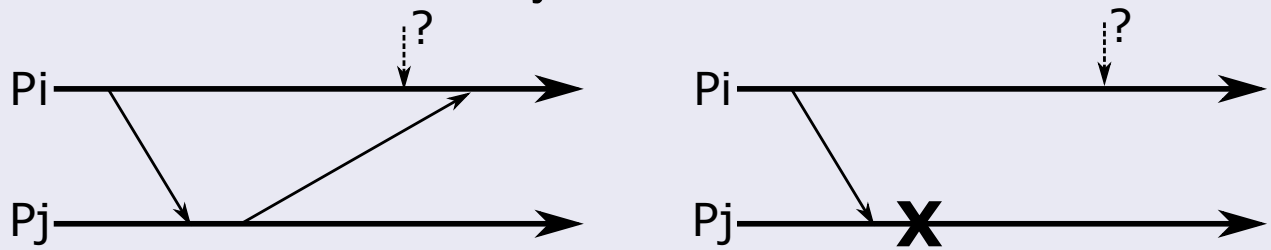
♪♪♪

Défaillance de la communication

Perte de message, modification du contenu, ordre de délivrance
⇒ solutions réseau ou algorithmiques

Défaillance de site

- arrêt du site, réponse erronée, transition erronée
- défaillance partielle du système
- non détectable en asynchrone : lent ou cassé ?



« *A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.* » Leslie Lamport, 1987.

Les épines, en résumé

♪♪♪

Impact de la répartition

- Pas d'horloge globale : chaque site a son horloge
- Pas d'état global immédiat accessible à un site
- Fiabilité partielle : possibilité d'arrêt d'une machine, d'un processus quel que part
- Sécurité relative : usagers potentiels nombreux
- Non déterminisme (parallélisme) : système asynchrone

Conséquence

Modèle de calcul différent du cas centralisé

- Ordre partiel entre les événements d'un calcul
- Calcul d'état global passé

Plan

Défis : point de vue système

- Mécanismes de communication : messages ou flux, RPC, RMI
- Processus : migration de code, démarrage/arrêt à distance
- Nommage : noms globaux à partir de noms locaux
- Synchronisation
- Stockage des données à distance (rapide, scalable)
- Réplication et cohérence
- Tolérance aux fautes : communication, nœud, processus
- Sécurité distribuée : canaux sécurisés, gestion des clefs, authentification et autorisations
- Passage à l'échelle
- Transparence

Défis : Algorithmique, conception - briques de base

- Modèles standard d'exécution : pour raisonner et concevoir des algorithmes répartis corrects
  - Modèle par entrelacement
  - Ordre partiel des événements
  - Logiques adaptées : TLA$^+$, IO Automata, π-calcul
- Algorithmes distribués sur les graphes
  - Topologie : graphe distribué avec connaissance partielle
  - Algorithmes de graphe : communication de groupe, diffusion de l'information, localisation d'objets
  - Graphes non structurés, dynamiques
  - Algorithmes efficaces : latence, trafic, mémoire locale, congestion
- État et temps global
  - Temps logique qui capture les dépendances entre événements
  - Calcul d'état global
  - Concurrence intrinsèque

Défis : Algorithmique, conception - coordination

- Mécanismes de synchronisation / coordination
  - Horloges physiques synchronisées
  - Élection de leader
  - Exclusion mutuelle
  - Interblocage distribué
  - Détection de la terminaison
  - Ramasse-miette : objet inaccessible de nulle part
- Communication de groupe, diffusion
  - Notion de groupe de processus
  - Aspects dynamiques : insertion, retrait volontaire ou pas, exclusion
  - Délivrance ordonnée
- Réplication de données
  - Modèles de cohérence : cohérence vs coût vs efficacité
  - Caches distribués pour accès rapide
  - Coordination de mises à jour
  - Placement optimal des copies

Défis : Algorithmique, conception - fiabilisation

- Surveillance et observation
  - Prédicats globaux
  - Mise au point distribuée (*debugging*)
  - Reconstruction de flux d'événements
- Outils de conception et de vérification d'algorithmes distribués
- Fiabilité et tolérance aux fautes
  - Algorithmes de consensus : accord en dépit de défaillance
  - Réplication
  - Algorithmes de votes
  - Bases de données distribuées
  - Systèmes auto-stabilisants : un état illégal conduit à un état légal
  - Point de sauvegarde (*checkpoint*) et algorithme de récupération (*recovery*)
  - Détecteur de défaillance : suspicion de défaillance

Défis : Algorithmique, conception - temps et efficacité

- Équilibrage de charge
  - Migration de calcul
  - Migration de données
  - Ordonnancement et allocation distribuée
- Temps-réel distribué : difficile en absence d'état et temps global
- Simulation distribuée

Défis : nouvelles applications

- Terminaux mobiles
  - Communication sans fil
  - Graphes dynamiques
  - Communication locale (modèle ad-hoc)
- Réseau de senseurs : petits processeurs à petite mémoire et faible communication (débit, distance)
- Informatique ubiquitaire
- Application pair-à-pair : rôle symétrique, absence de hiérarchie, auto-organisant, dynamique
- Diffusion de contenu à grande échelle
- *Data mining* : données provenant de plusieurs dépôts répartis
- *Grid computing*
- Sécurité, *privacy*

Plan

Principe de conception

Une idée clé : la transparence

Principe de conception 1

Un bon système réparti
est un système
qui semble centralisé
(qui s'utilise comme)

Principe de conception 2

Un bon système réparti
n'est pas un système centralisé

[Précis 1.5 pp.13–18]

Idée : masquer la répartition

Niveaux de transparence

- Accès
- Localisation
- Partage
- Réplication
- Fautes
- Migration
- Charge
- Échelle

Mécanismes

- Interface
- Nommage
- Synchronisation
- Groupe
- Atomicité
- Mobilité
- Réflexivité
- Reconfiguration

Transparence d'accès

Propriété

Accès à une ressource distante $\equiv$ accès à une ressource locale

Exemple

- Niveau langage de commande : sh $\neq$ ssh (non transparence)
- Niveau service système : read,write identiques que le fichier opérande soit local ou distant (transparence)
- Niveau langage à objet : appel de méthode local ou à distance identique pour l'appelant (transparence)

Solution : Notion d'interface

Cas des intergiciels à objets : langage IDL et bus logiciel

Transparence de localisation

Propriété

La localisation d'une ressource reste cachée.

Exemple

- Non transparence : commande scp bach.enseeiht.fr:/foo .
- Transparence :
  - Niveau service système : open("nom-fichier",...) : nom du fichier indépendant de la localisation du fichier
  - Niveau langage à objet : références aux objets distants sans nécessité de connaître leur localisation

Solution : Services de nommage gérant des noms globaux

Cas des intergiciels à objets : serveurs de noms

Transparence du partage

Propriété

L'usage partagé (et en parallèle) d'une ressource doit rester cohérent ($\equiv$ sémantique équivalente au cas centralisé).

Exemple

- Niveau service système : cohérence d'accès à un fichier partagé : assurer les contraintes d'exclusion mutuelle des lecteurs/rédacteurs, mais coûteux
- Niveau langage à objets : limiter l'exécution en parallèle des méthodes sur un objet

Solution : Mécanismes de synchronisation

Problème : mécanismes connus mais souvent coûteux en réparti

Transparence de la réplication

Propriété

La répartition permet la redondance pour plus de fiabilité

Exemple

- Niveau service système : assurer le maintien de plusieurs copies cohérentes d'un même fichier
- Niveau langage à objets : assurer la réplication transparente d'un objet
- Niveau intergiciel : assurer que plusieurs serveurs répliqués évoluent en cohérence

Solution : Synchronisme virtuel

Notion de groupe et de protocoles de diffusion atomique

Transparence des fautes

Propriété

La répartition induit un contexte moins fiable que celui du centralisé : panne partielle

Exemple

- Niveau service système : un service n'est plus accessible (serveur de noms !)
- Niveau langage à objets : un appel à distance de méthode peut échouer

Solution : Traitement d'exception et atomicité

Atomicité : un traitement s'exécute en entier ou pas du tout

Transparence de la migration

Propriété

Permettre la migration de code, de processus, d'agents, d'objets.

Exemple

- Niveau service système : déplacer un serveur d'une machine chargée à une machine sous-utilisée
- Niveau langage à objets :
  - code mobile : langages de script
  - objets mobiles (ou agents mobiles)

Solution : la mobilité des traitements et/ou des données

Agents mobiles (contexte d'exécution mobile), code mobile, machine virtuelle

Transparence de charge

Propriété

Masquer (et empêcher) les phénomènes de surcharge, écroulement

Exemple

La répartition permet naturellement la mise en œuvre de techniques d'équilibrage de charge

- Niveau système : reconfigurer dynamiquement les services sur les machines disponibles selon la charge des serveurs
- Niveau grappe (cluster) : répartir les traitements parallèles de façon équilibrée sur les différents processeurs

Solutions : réflexivité, machine virtuelle

Réflexivité : possibilité d'auto observation des composants
Machine virtuelle : dissocier environnement d'exécution et support matériel

Transparence d'échelle

Propriété

Permettre l'extension d'un système sans remettre en cause son fonctionnement global

Exemple

- Niveau système : introduire de nouveaux serveurs sur de nouvelles machines pour s'adapter à une augmentation de l'activité applicative

Solution : Adaptabilité et autonomie

Adaptabilité et autonomie : mise en œuvre de mécanismes automatique d'adaptation dynamique

En résumé

Répartition
≡
Accès et partage de ressources
via un réseau de communication
à tout usager qui en a le droit
et où qu'il soit

Les épines

- Pas d'horloge globale
- Pas d'état global immédiat
- Fiabilité partielle
- Sécurité relative
- Non déterminisme