

Systèmes et algorithmes répartis

Principes et concepts

Philippe Quéinnec, Gérard Padiou

queinnec@enseeiht.fr

<http://queinnec.perso.enseeiht.fr/Ens/sar.html>

ENSEEIHT

Département Sciences du Numérique

30 septembre 2021

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

Plan

- 1 Préambule
- 2 Définition et problématique
 - Les parfums
 - Exemple
 - Les épines
- 3 Un principe de conception : la transparence



Sources

- G. Padiou, « *Précis de répartition* », 2016,
<http://queinnec.perso.enseeiht.fr/Ens/SAR/precis.pdf>
Référéncé 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>



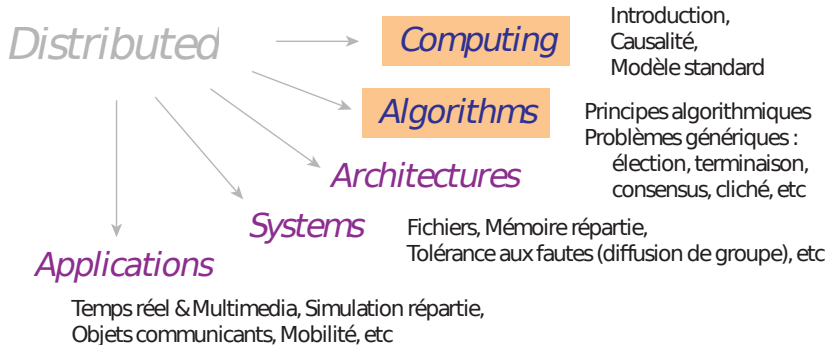
Préambule : tendance

Répartition \equiv communication entre objets informatisés

- Actuellement : ordinateurs + téléphones + tablettes toujours connectés
- L'Internet des objets (*The Internet of things*)
 - 24 milliards d'appareils connectés entre eux 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



Plan

- 1 Préambule
- 2 Définition et problématique
 - Les parfums
 - Exemple
 - Les épines
- 3 Un principe de conception : la transparence



Modèle centralisé ou réparti



Modèle centralisé

Les processus se partagent
des ressources critiques ou pas



Modèle réparti

Les processus échangent
des données par messages



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]



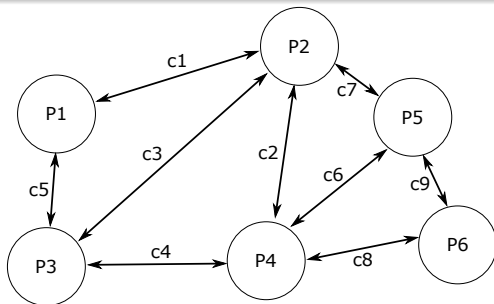
Exemple : découvrir le graphe de communication



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 $voisins_i = \{id_x, id_y, \dots\}$.

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.



```

on start :
  for each  $id_j \in voisins_i$  do
    send  $\langle id_j, voisins_i \rangle$  to  $id_j$ 
  end for
   $sites\_known_i \leftarrow \{id_i\}$ 
   $channels\_known_i \leftarrow$ 
    {  $(id_j, id_k) : id_k \in voisins_i$  }
    
```

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 reception  $\langle id, voisins \rangle$  :
  if  $sites\_known_i = \emptyset$  then start(); fi
  if  $id \notin sites\_known_i$  then          -- premier message de id
     $sites\_known_i \leftarrow sites\_known_i \cup \{id\}$ 
     $channels\_known_i \leftarrow channels\_known_i \cup \{ (id, id_k) : id_k \in voisins \}$ 
    for each  $id_j \in voisins$           -- propage le message
      send  $\langle id, voisins \rangle$  to  $id_j$ 
    end for
    if  $\forall (id_j, id_k) \in channels\_known_i : \{id_j, id_k\} \subseteq sites\_known_i$  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 ? ($2 * \text{nombre de sites} * \text{nombre de canaux}$)
- Complexité en temps ? ($2 * \text{diamètre}$)
(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



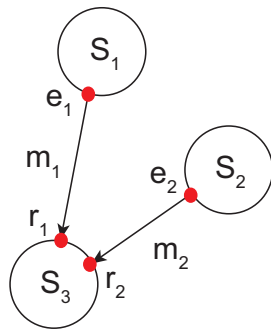
Problèmes...

- m_1 est-il toujours envoyé avant m_2 dans toute exécution ?
- m_1 est-il toujours reçu avant m_2 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



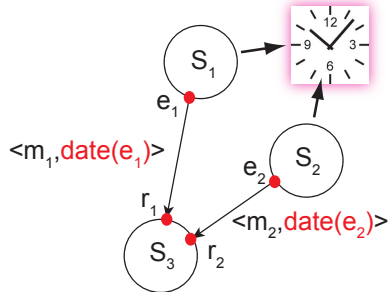
Dates dans messages

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

⇓ ?

e_1 avant ? e_2

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



Épine : pas de temps global

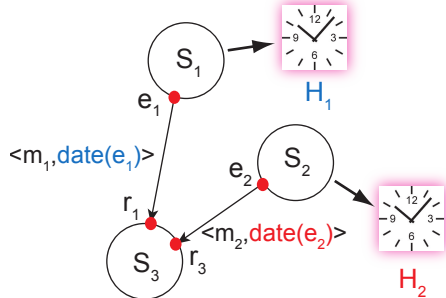
il existe 2 horloges

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



e_1 avant e_2

Si les horloges
sont synchronisées !



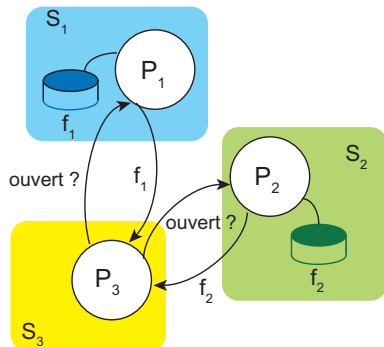
Pas de référentiel temporel unique

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

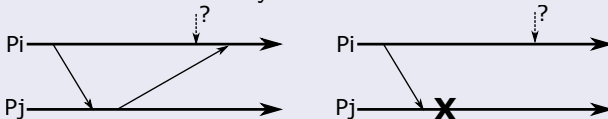


Défaillance de la communication

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



Thèmes de recherche sur la répartition

Concevoir, modéliser, expérimenter

- Modélisation théorique
- Algorithmique
- Langages
- Systèmes d'exploitation
- Intergiciels (middleware)

[Précis 1.4 pp.11–12]



Plan

- 1 Préambule
- 2 Définition et problématique
 - Les parfums
 - Exemple
 - Les épines
- 3 Un principe de conception : la transparence



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



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

