Algorithmique répartie

Problèmes génériques : prise de cliché et consensus

Gérard Padiou

Département Informatique et Mathématiques appliquées
ENSEEIHT

Octobre 2012

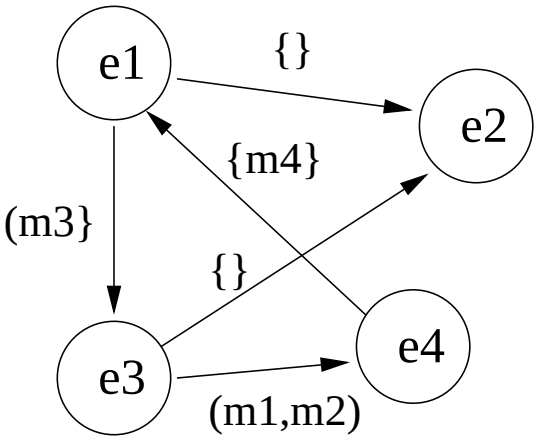
plan

Plan

Prise de cliché (snapshot)

Definition

Objectif : Capter, centraliser un état global passé des processus



- Prise instantanée impossible ;
- Un site collecteur accumule ;
- Prise cohérente de clichés locaux ;
- Problème des messages en transit ;

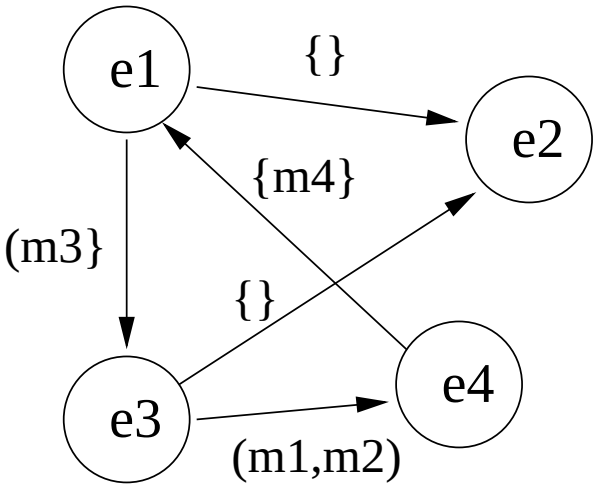
Cliché global instantané

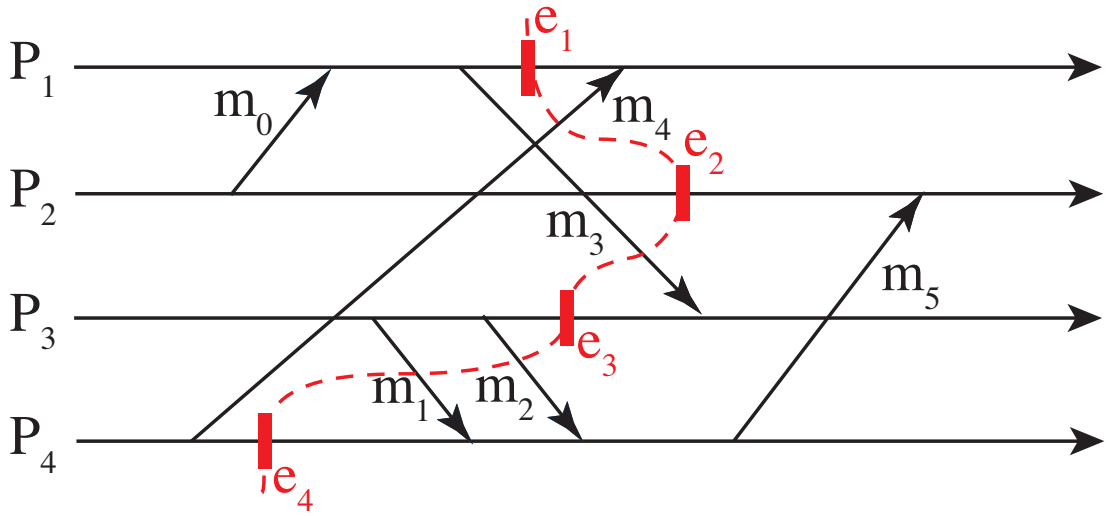
Clichés locaux + Messages en transit

$$\{e_1, e_2, e_3, e_4\} + \{m_1, m_2, m_3, m_4\}$$

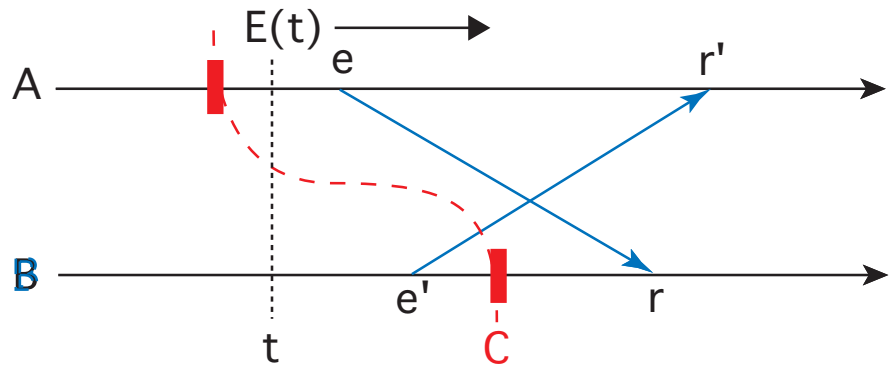
Prise de cliché (snapshot)

Schéma temporel de la prise de cliché





Virtualité du cliché pris

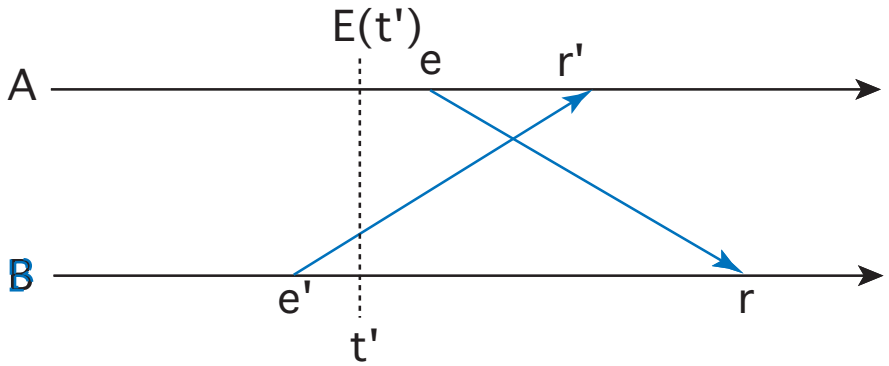


Réalité

Le cliché pris est cohérent mais ...

Cliché effectif

Il n'a pas existé à un instant global



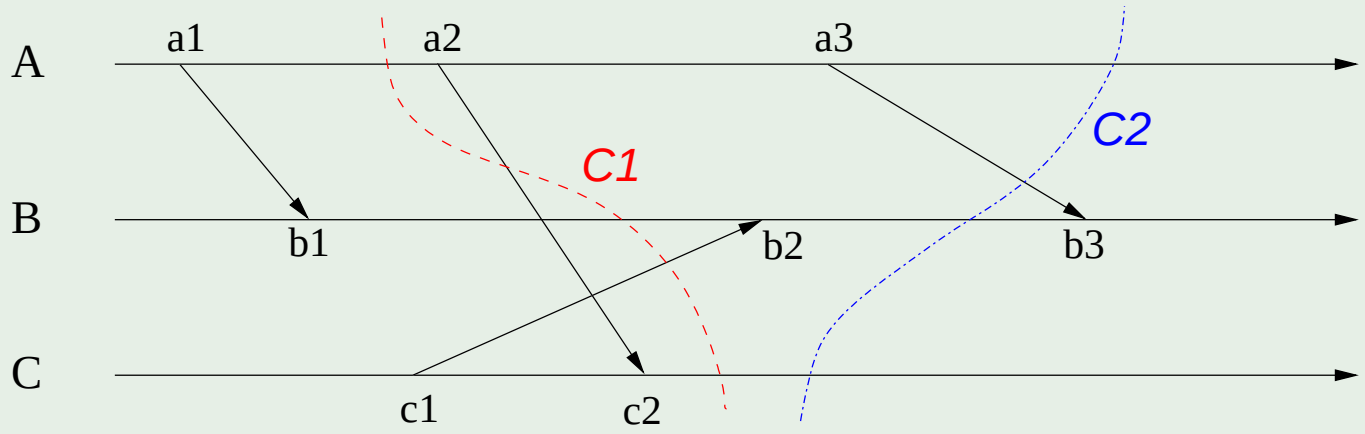
Cohérence d'un cliché

Notion de coupe

- Idée : Délimiter un passé et un futur d'une exécution répartie.
- Coupe C = ensemble d'événements « passés »
- Coupe cohérente $\equiv \forall e \in C : \forall e' : e' \prec e \Rightarrow e' \in C$

Exemple

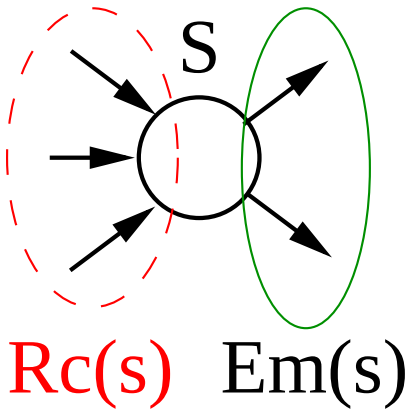
C1 : incohérente, *C2* cohérente



Algorithme de Misra-Chandy

Hypothèses

- Réseau fortement connexe ($\forall s, s' : \exists s \rightarrow s'$)
- Canaux unidirectionnels et fifo :
  $\forall s$, $Rc(s)$: canaux en réception
  $Em(s)$: canaux en émission



Principes de l'algorithme

- Utilisation de messages marqueurs
- Répartition de l'évaluation : chaque site évalue :
  - son cliché local ;
  - Les messages en transit sur ses canaux en réception : $\forall c \in Rc(s)$

Algorithme de Misra-Chandy

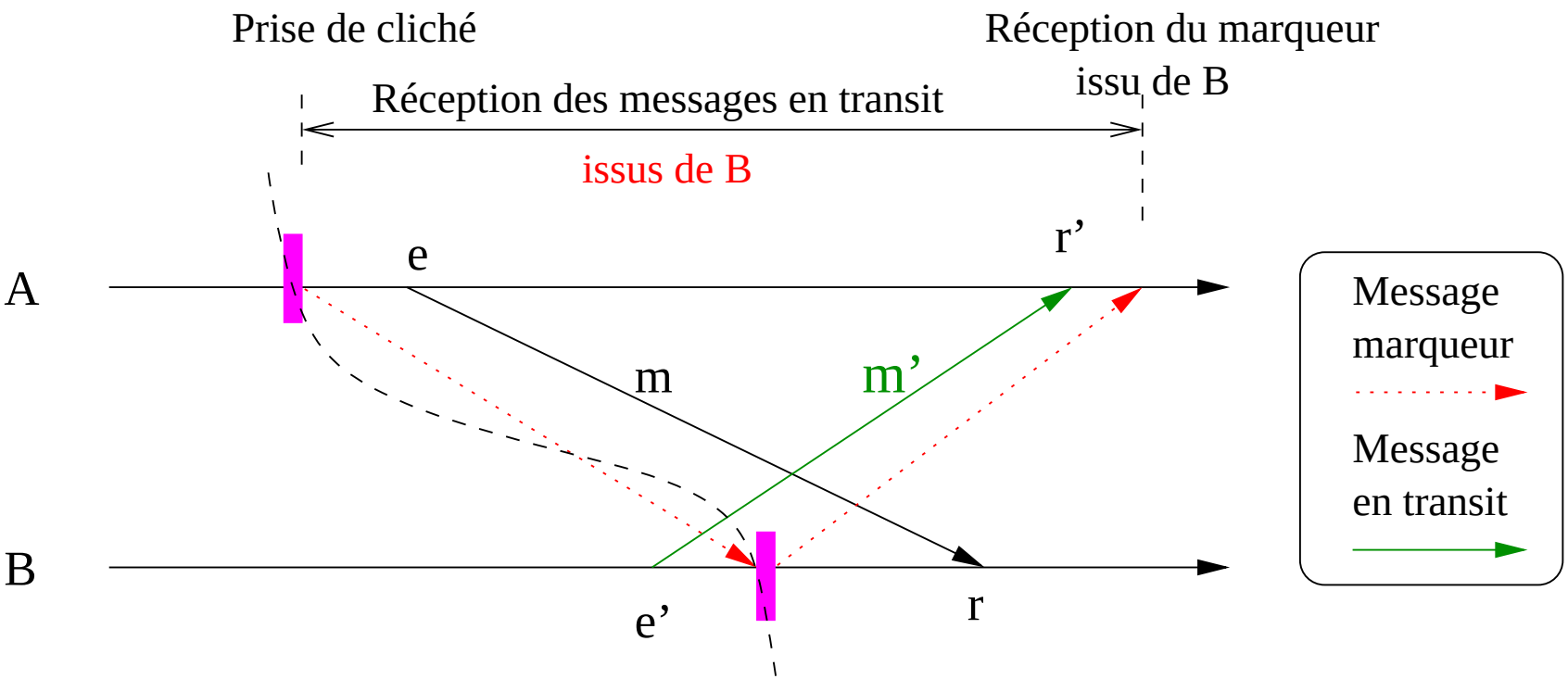
Comportement d'un site

Sur réception d'un PREMIER marqueur ou spontanément :

1. Prendre son cliché local L_s et émettre un marqueur sur chaque canal d'émission $c \in Em(s)$
2. Enregistrer dans une liste $enTransit[c]$ les messages reçus sur chaque canal de réception $c \in Rc(s)$ jusqu'à la réception d'un marqueur
3. Lorsqu'un marqueur a été reçu sur TOUS les canaux de réception, communiquer au collecteur cet état partiel :

$$\{L_s, enTransit[c], \forall c \in Rc(s)\}$$

Prises de clichés locaux et marqueurs



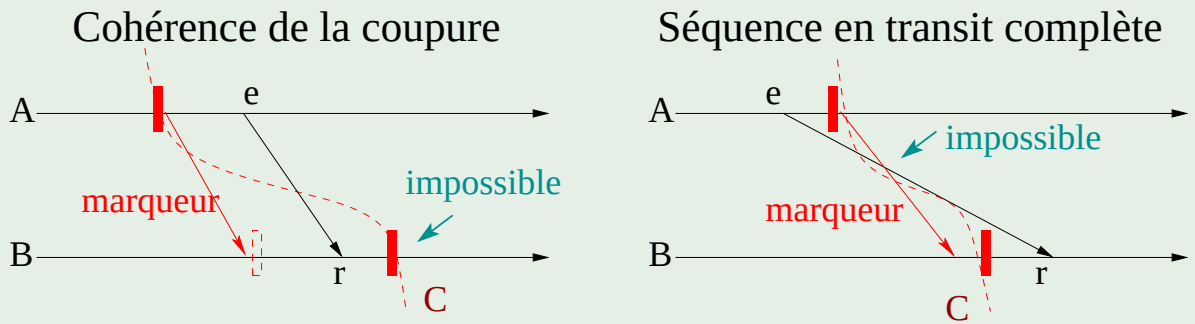
Rappel : l'arrivée du marqueur => prise de cliché

A vérifier...

Propriétés

- Sûreté
  - Coupure cohérente
  - Collecte complète des messages en transit
- Vivacité
  - Tout site finit par prendre un cliché local
  - Un marqueur finit par arriver sur chaque canal de réception

Exemple



Reprise d'un calcul réparti à l'aise de points de reprise

Problème voisin de la prise de cliché

Le problème

Suite à l'arrêt d'un site :

☞ redémarrer le site en alertant les autres sites pour une reprise globale du calcul à partir d'un état passé cohérent.

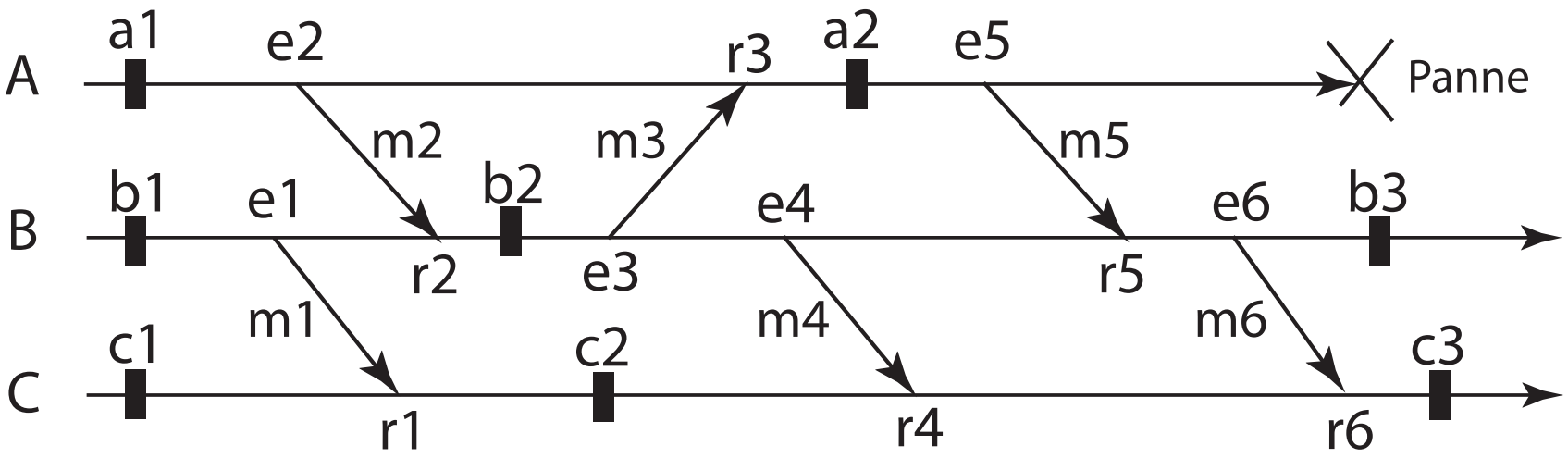
Contextes d'usage : calcul scientifique, systèmes transactionnels

Quelques questions

- Qu'est-ce qu'un état cohérent ?
- Comment placer des points de reprises locaux pour reprendre le calcul dans un état le plus récent possible ?
- Que faut-il rejouer au point de vue échange de messages ?

Reprise d'un calcul réparti

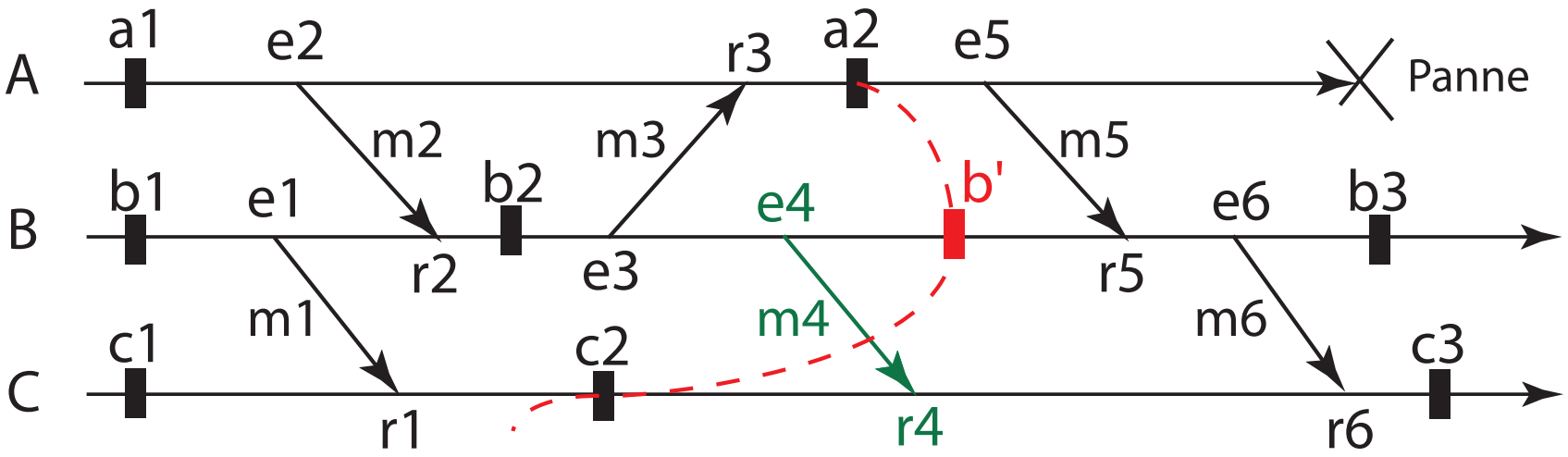
Une difficulté (connue en centralisé aussi) : l'effet domino



- choisir a_2, mais b_3 impossible, donc revenir à b_2 ;
- choisir b_2 mais alors a_2 impossible, donc revenir à a_1 ;
- choisir a_1 mais alors b_2 impossible, donc revenir à b_1 ;
- choisir b_1 mais alors c_2 et c_3 impossibles, donc revenir à c_1.

Reprise d'un calcul réparti

Une difficulté (connue en centralisé aussi) : l'effet domino



- Une sauvegarde b' évite l'effet domino ;
- Attention : un message en transit (m_4) devra être « rejoué »
- Comment bien coordonner les points de reprise locaux ?
- Comment minimiser le nombre de points de reprises locaux ?
- Comment traiter les interactions avec l'extérieur ?

Reprise d'un calcul réparti

Checkpoint-based rollback recovery

Coordination des points de reprise locaux

Déclenchement périodique par un site initiateur :

☺ un seul « dernier point de reprise » à garder ;

☺ pas d'effet domino ;

☞ **Protocole bloquant à deux phases :**

☹ délais sur les interactions extérieures ;

☹ délais dus aux phases de prise de « cliché de reprise »

☞ **Protocole non bloquant : clichés à la Misra/Chandy :**

☺ une seule phase et pas de blocage ;

☹ $\Rightarrow$ canaux FIFO.

Reprise d'un calcul réparti

Checkpoint-based rollback recovery

Coordination des points de reprise locaux

Sauvegarde adaptative (Xu et Netzer)

☺ éviter l'effet domino ;

☹ points de reprises « forcés » ;

☞ **Principes algorithmiques :**

- Traquer les « zigzags » (la causalité) entre points » ;
- Forcer des points de reprises à bon escient (le - possible) ;

Une autre approche : Log-based rollback recovery

Principe : par journalisation des événements asynchrones : réception de messages, interruptions, etc

Plan

Origine et usage ($\rightarrow$ Tolérance aux fautes)

Etude approfondie par S. Krakowiak :
`http://proton.inrialpes.fr/~krakowia/Enseignement/M2R-SL/SR/`

– Objectif –

Un groupe de processus finit par prendre la même décision

Usage

- Validation d'une transaction répartie
- Diffusion d'un message
- Connaissance d'une défaillance d'un site
- Accord sur le résultat d'un calcul répliqué
- ...

ATTENTION : Multiples solutions selon les hypothèses faites sur le contexte d'exécution : (a)synchrone, mémoire partagée ou messages, processus byzantins ...

Origine et usage ($\rightarrow$ Tolérance aux fautes)

Etude approfondie par S. Krakowiak :
`http://proton.inrialpes.fr/~krakowia/Enseignement/M2R-SL/SR/`

– Objectif –

Un groupe de processus finit par prendre la même décision

Usage

- Validation d'une transaction répartie
- Diffusion d'un message
- Connaissance d'une défaillance d'un site
- Accord sur le résultat d'un calcul répliqué
- ...

ATTENTION : Multiples solutions selon les hypothèses faites sur le contexte d'exécution : (a)synchrone, mémoire partagée ou messages, processus byzantins ...

Spécification

- Système composé de N processus
- Un domaine de valeur $\mathcal{D}$
- Chaque processus possède une valeur initiale $v_0 \in \mathcal{D}$
- Tous les processus évaluent une même valeur finale v_f
- Variantes : contraintes de validité de la valeur finale choisie

Exemple

Spécification

- Système composé de N processus
- Un domaine de valeur $\mathcal{D}$
- Chaque processus possède une valeur initiale $v_0 \in \mathcal{D}$
- Tous les processus évaluent une même valeur finale v_f
- Variantes : contraintes de validité de la valeur finale choisie

Exemple

Valeur initiale	True	False	True	True	True
Valeur finale	False	False	False	False	False

Spécification (suite)

Contraintes de validité

- La valeur finale ne peut être que l'une des valeurs initiales :
$$v_f \in \{v_0^1, \ldots, v_0^N\}$$
- Corollaire : Si tous ont la même valeur initiale, la valeur finale ne peut être que cette valeur commune :
$$(\forall i \ : \ v_0^i = v) \ \Rightarrow \ v_f = v$$

Solutions triviales si contexte d'exécution fiable
Solutions complexes (voire pas du tout de solution) dès qu'une faute peut survenir

Spécification (suite)

Contraintes de validité

- La valeur finale ne peut être que l'une des valeurs initiales :
$$v_f \in \{v_0^1, \ldots, v_0^N\}$$
- Corollaire : Si tous ont la même valeur initiale, la valeur finale ne peut être que cette valeur commune :
$$(\forall i \ : \ v_0^i = v) \ \Rightarrow \ v_f = v$$

Solutions triviales si contexte d'exécution fiable

Solutions complexes (voire pas du tout de solution) dès qu'une faute peut survenir

Spécification (suite)

Contraintes de validité

- La valeur finale ne peut être que l'une des valeurs initiales :
$$v_f \in \{v_0^1, \ldots, v_0^N\}$$
- Corollaire : Si tous ont la même valeur initiale, la valeur finale ne peut être que cette valeur commune :
$$(\forall i \ : \ v_0^i = v) \ \Rightarrow \ v_f = v$$

Solutions trivales si contexte d'exécution fiable
Solutions complexes (voire pas du tout de solution) dès qu'une faute peut survenir

Une classification

Résultat d'impossibilité de M. Fisher, N. Lynch et M. Paterson (1985)

Processus communiquant par mémoire partagée

Cas	Défaillance de processus
synchrone	$\exists$ solution
asynchrone	pas de solution

Processus communiquant par messages

Cas	Défaillance de processus	Perte de message
synchrone	$\exists$ solution	pas de solution
asynchrone	pas de solution	pas de solution

Introduction des détecteurs de fautes

T.D. Chandra, S. Toueg, Unreliable failure detectors for reliable distributed systems, Journal of the ACM, vol. 43, n°2, 1991

Definition

Idée : Associer à chaque processus un module appelé détecteur de faute qui gère une liste des processus « suspects » (≡ considérés comme arrêtés).

Il existe des algorithmes sous ces nouvelles conditions.

Complétude (« connaître tous les suspects »)

- Faible : Finalement tout processus défaillant est suspecté de façon permanente par au moins un processus correct
- Forte : Finalement tout processus défaillant est suspecté de façon permanente par tous les processus corrects

Propriétés des détecteurs de fautes

Exactitude (toujours)

- Faible : Au moins un processus correct n'est jamais suspecté
- Forte : Aucun processus n'est suspecté avant son arrêt.

Exactitude(finalement)

- Faible : Il existe une date à partir de laquelle au moins un processus correct n'est pas suspecté par tous les autres processus corrects
- Forte : Il existe une date à partir de laquelle aucun processus correct n'est suspecté par tous les autres processus corrects

Classes de détecteurs de fautes

complétude	exactitude			
	toujours		finalement	
	forte	faible	forte	faible
forte	$\mathcal{P}$(erfect)	$\mathcal{S}$(trong)	$\Diamond\mathcal{P}$	$\Diamond\mathcal{S}$
faible	$\mathcal{Q}$	$\mathcal{W}$(eak)	$\Diamond\mathcal{Q}$	$\Diamond\mathcal{W}$

Remarque

La complétude faible suffit, puisqu'une fois qu'un processus correct possède une connaissance exacte des suspects, il peut la diffuser à tous les processus corrects.

Une solution « probabiliste » au consensus en asynchrone

Michael Ben-Or, Another Advantage of Free Choice : Completely Asynchronous Agreement Protocols, 2nd ACM Symposium on Principles of Distributed Computing, 1983

Premier exemple de problème de synchronisation qui a une solution probabiliste mais pas de solution déterministe.

Principe

- Si tout le monde est d'accord (propose la même valeur) alors cette valeur est choisie ;
- Sinon, la valeur finale résultera d'un choix probabiliste (et non pas déterministe).

☞ Sûreté : Terminaison assurée

Algorithme probabiliste

```
process P(i :0..N-1) {
 /* T = nombre maximum de défaillances < N/2 */
 int r = 1 ; boolean b=b0 ; boolean x;
 int nbv, nbf; boolean Décidé = False;
 while (true) {
  /* -r.1- Propagation des valeurs initiales */
  send Proposition(r,b) to P(0..N-1);
  nbv=0; nbf=0;
  for(int i=0 ; i<N-T; i++) { /* réception de ces valeurs */
    receive Proposition(r,x); if (x) nbv++ ; else nbf++ ;
  }
  /* -r.2- Propagation des choix de chaque processus */
  if(nbv>N/2) send Choix(r,True) to P(0..N-1);
  else if(nbf>N/2) send Choix(r,False) to P(0..N-1);
  else send Choix(r,?) to P(0..N-1);
```

Algorithme probabiliste (suite)

```
  int k=0; /*  -r.3- Collecte des choix */
  for(int i=0; i<N-T; i++) {
    select { /* les b's reçus ont même valeur */
      when receive Choix(r,x) ; k++;
    ||
      when receive Choix(r,?);
    }
  }
  /* -r.4- Décider ou pas */
  if (k==0) b=Random(True,False); else b=x;
  Décidé = (k>T) ; /* ⇒ tous voient k > 0 */
  r=r+1; /* Passer au pas suivant */
 } // while
}
```

Algorithme probabiliste (suite)

Propriétés

Si $N > 2T$, alors l'algorithme garantit avec une probabilité de 1 que :

- *Terminaison* : Tous les processus décideront finalement de la même valeur ;
- *Intégrité* : Si tous les processus proposent la même valeur initiale, alors ils décident de cette valeur en un seul tour ;
- *Vivacité* : Si, pour un tour r_0, un processus décide une valeur b_f, alors tous les autres processus décideront la même valeur b_f lors du tour suivant $r_0 + 1$.

Inconvénient. . .

Caractère probabiliste

- Lorsque les processus ne sont pas d'accord initialement, la valeur finale dépend de l'exécution (non déterminisme) ;
- Conséquence : un même état initial peut conduire selon l'exécution considérée à un résultat final différent ;
- Conclusion : pas utile lorsque l'on veut un résultat déterministe si les processus ne sont pas initialement d'accord.
  ☞ Cas de la validation d'une transaction par exemple

Plan

Une première étape : la diffusion fiable

Hypothèses

- communication asynchrone fiable ($\forall m : e(m) \mapsto r(m)$)
- mais défaillance par arrêt des processus.

Propriétés

- Atomicité : Si un processus correct délivre m, tous les processus corrects délivrent m
- Validité :Si un processus correct diffuse m, tous les processus corrects délivrent m
- Intégrité : $\forall m$, m est délivré une seule fois à tout processus correct ssi il a été diffusé par un processus.

Une première étape : la diffusion fiable

Implantation du broadcast

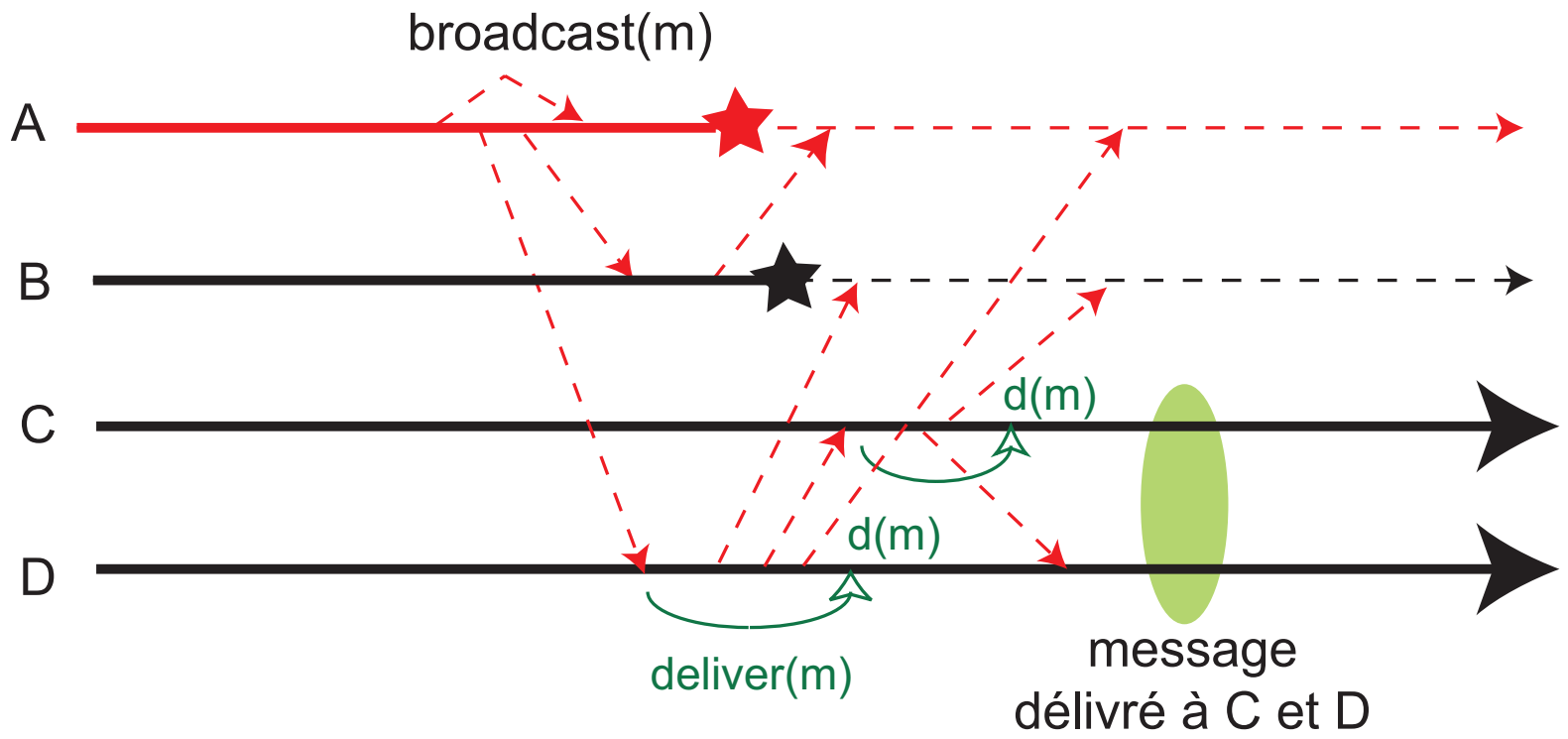
```
void broadcast(émetteur p, message m) {
  for (dest ∈ voisins(p)∪ p) {send(<m,seq(m)>,dest)};
  // occurrences de e_p(m)
}
```

Implantation du deliver pour un processus p

```
upon receive(m) { // occurrence de r_p(m)
  if (¬ delivered(p,m)) {
    for (dest ∈ voisins(p)) {send(<m,seq(m)>,dest)};
    deliver(m) ; // occurrence de d_p(m)
  }
}
```

Diffusion fiable

Exemple



Un algorithme « partiel » Paxos

L. Lamport, The part-time parliament, ACM Transactions on Computer Systems, vol. 16, n°2, May 1998, pp.133-169

Idée

- Algorithme non vivace : peut retarder la décision indéfiniment ;
- Lorsque le consensus est atteint, il est correct.
- Utilité : conditions de vivacité vérifiées « la plupart du temps »

Hypothèses

- Communication :
  - asynchrone,
  - pas d'altération de messages,
  - mais possibilité de perte de messages
- Nombre fixe de processus
- Défaillance par arrêt avec possibilité de reprise.

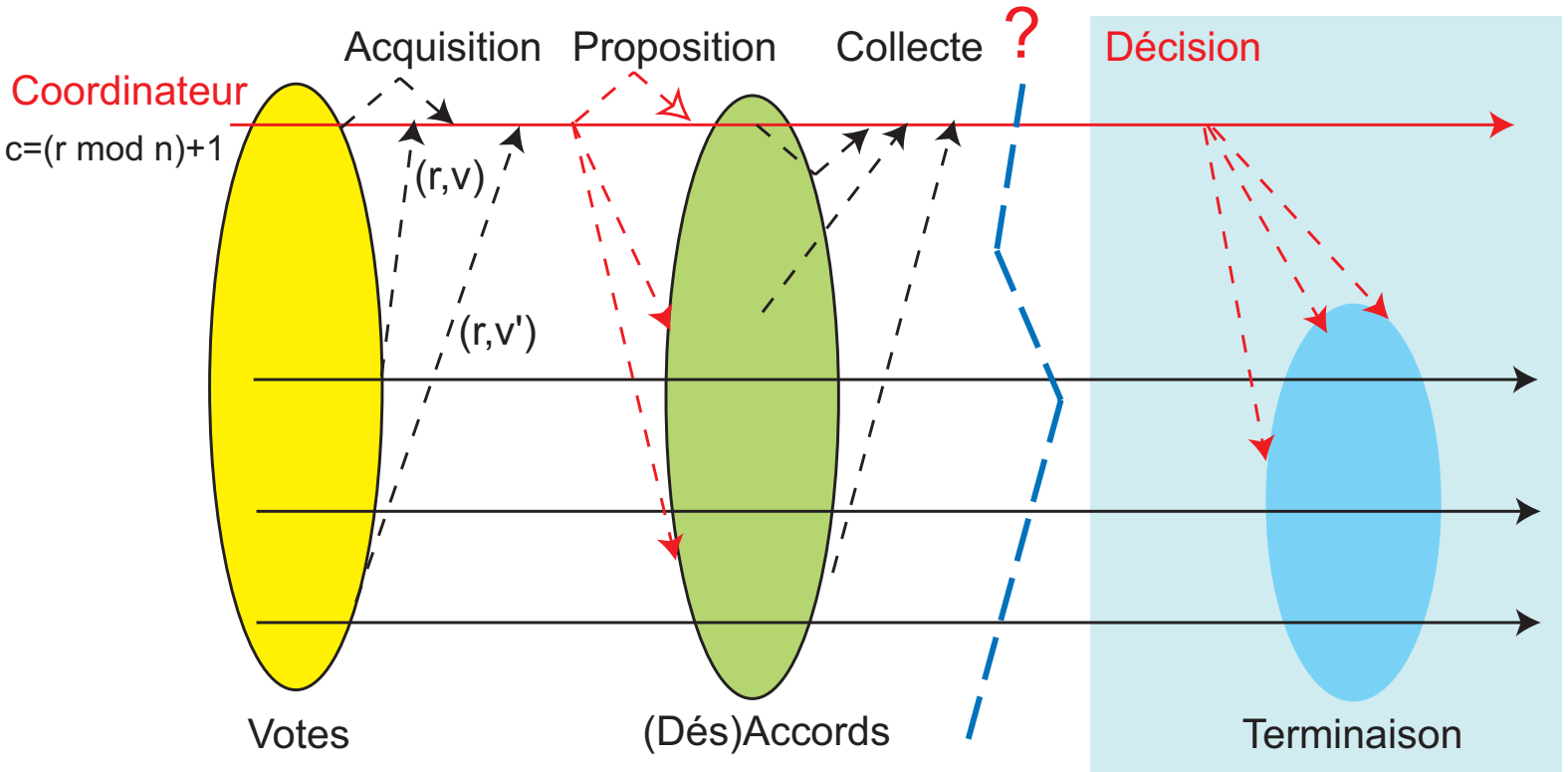
Un algorithme avec détecteur de type $\diamond S$

Principe de l'algorithme

- Utilisation de la notion de tours (rounds) ;
- Un coordinateur unique est associé à chaque tour
  $\rightarrow$ envoi spontané vers le coordinateur du tour courant ;
- Le coordinateur « change » à chaque tour ;
- Un coordinateur décide ou passe au tour suivant ;
- Un participant :
  - vote,
  - accepte ou refuse une proposition,
  - accepte la décision

Un algorithme avec détecteur de type $\diamond S$

Décomposition en phases d'un tour



La décision n'a lieu que si une majorité de processus est d'accord

```
processus P_i { // -- rôle coordinateur en violet --
  r = 0; r_v = 0 ; v = v_0 ; décidé = false ;
  while (¬ décidé) {
   c=(r mod N)+1;send(<vote,r,r_v,v>,c) ; // -1- vote auprès du coordinateur
   if (i==c) { // -2- le coordinateur collecte les votes
     while (votes < ⌈(n+1)/2⌉) { receive(<vote,r,r_x, v_x>); }
     rmax = max(r_x); v = v_rmax ;
     ∀d ≠ c : send(<prop,r,v>,d); // diffusion de la proposition
   }
   select { // -3- répondre à la proposition ou diffuser la décision
     receive(prop,r,x) ⇒{ v=x; r_v=r; send(<ack>,c); }
   or // coordinateur suspecté
      p_c ∈ Suspects_i ⇒ send(<nack>,c);
   or // diffusion fiable de la décision (causée par le coordinateur)
     receive(decide,v_f) ⇒{ ∀d : send(<decide,v_f>,d); décidé = true ; }
   }
   if (i==c) { // le coordinateur décide ou non
     while (réponses nack|ack's < ⌈(n+1)/2⌉) { receive(nack|ack); }
     if (tous d'accord) { ∀d : send(<decide,v>,d) ; }
   }
   r++; //  passage au tour suivant
  }
}
```

Un algorithme avec détecteur de type $\diamond S$

Propriétés

- Tolérance :
  tolérer f arrêts parmi N processus $\Rightarrow 2f + 1 \leq N$;
- Majorité :
  un coordinateur correct réunit finalement une majorité ;
- Vivacité
  $\diamond S \Rightarrow$ un coordinateur correct, non soupçonné décide.

La décision est diffusée aux processus corrects
selon le schéma algorithmique de diffusion fiable

Détecteurs de défaillances

Aspects pratiques

Problème ! ! !

Les détecteurs permettent de résoudre le problème du consensus en asynchrone : ils sont donc irréalisables (Conséquence de FLP).

Solution : lever l'hypothèse d'asynchronisme (Ouf ! ! !)

Introduire un délai de garde pour détecter l'arrêt d'un processus.
$\Rightarrow$ borne supérieure estimée Δ sur la transmission d'un message.

Mise en œuvre

- Test périodique par *ping* : envoi périodique d'un *ping* qui doit être acquitté dans un délai de 2Δ ;
- Diffusion périodique de *alive* : diffusion périodique d'un *alive* ($\Rightarrow$ synchronisation des horloges physiques).