



Rejeu d'un calcul d'agents mobiles

Mamoun Filali, Philippe Mauran, Gérard Padiou, Philippe Quéinnec

FAC'02

LAAS

26-27 mars 2002

Plan

- Agents mobiles
- Vecteurs de chemin, encodage d'un arbre d'exécution
- Rejouer une exécution
- Reconstruction

Modélisation d'un calcul d'agents mobiles

Contexte : Système distribué

Ensemble (dynamique) de sites, messages fiables asynchrones.

Agent mobile

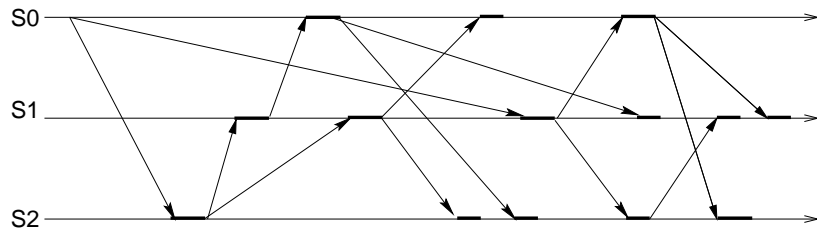
Processus qui peut :

- migrer d'un site vers un autre (**go**), i.e. suspendre son exécution, transférer son code et ses données vers le site de destination D, et reprendre l'exécution sur le site D ;
- sur un site :
 - faire du travail local
 - créer de nouveaux agents (**fork**), qui migrent vers d'autres sites.

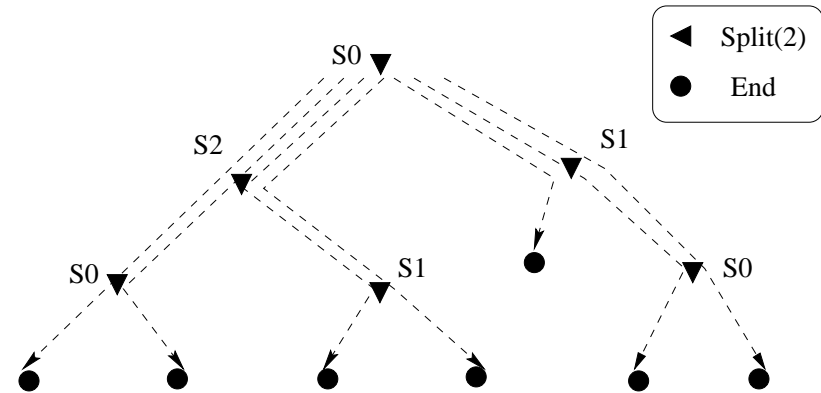
Schéma d'un agent mobile

```
AgentModel() {  
    while (true) {  
        // L'agent arrive sur un site  
        // Exécution locale sur le nœud courant  
        SetOfUrl dest = {... };  
        if (dest.isEmpty()) exit(); // Fin de l'agent  
        // Migrations vers d'autres sites  
        Url s = fork(dest);  
        go(s); // Exécuté par chaque clone  
    }  
}
```

vue messages



vue contrôle

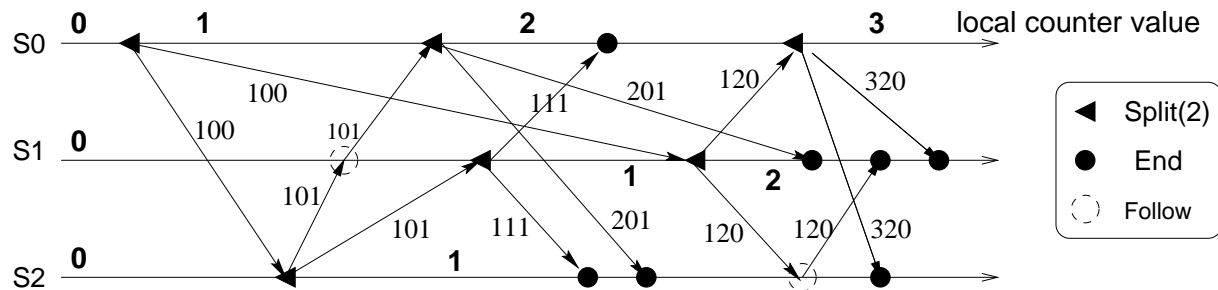


Un calcul mobile peut être vu comme un calcul diffusant.

Un chemin = séquence maximale de visites de la racine à une feuille
(dernière visite d'un agent)

Protocole

- chaque site s conserve (dans un compteur local $lc[s]$) le nombre de chemins créés sur ce site s ;
- chaque agent transporte un vecteur v ;
- À la réception d'un agent (a, v) , un site s peut :
 - créer $(d - 1)$ nouveaux chemins: le compteur local est incrémenté ($lc[s] := lc[s] + d - 1$), le vecteur de chemin est mis à jour, et d nouveaux messages sont émis,
 - ou terminer le chemin, et envoyer v à un site collecteur.



Modélisation du protocole

```
const N : posnat
type Site = 0..N-1
  path_vector=Site × array Site of nat -- vecteur de chemin
Program basic_path_vectors
declare st: set of path_vector -- ensemble des vecteurs
always
  ...
initially st = { (0, (0, ...)) }
assign -- sur le site s, split atomique du vecteur (s, V) sur les sites de D
  ⟨ ∥ (s, V, D) : g(s, V, D) : st := st \ {(s, V)} ∪ nv(s, V, D) ⟩
end
```

Rejeu d'une exécution distribuée

- Rejeu avant : refaire exécuter un calcul à "l'identique" : mêmes migrations dans le même ordre sur chaque site.
Objectif : même arbre de calcul.
- Rejeu arrière : annulation d'une étape du calcul (undo).

Utilité :

- debug
- traçage
- reprise d'un calcul

Cadre

- communications asynchrones
- jeu post-mortem : après terminaison du calcul.
- actions applicatives :
 - *déterministes*, basées uniquement sur l'état de l'agent et l'état du site.
 - ou non-déterministes, avec sauvegarde de chaque action sur chaque site
- ordre respecté = ordre causal sur les événements de migration :
 - e et e' sur le même site et e avant $e' \rightarrow e \prec e'$
 - e et e' sur le même chemin et e avant $e' \rightarrow e \prec e'$
 - fermeture transitive : $e \prec e' \wedge e' \prec e'' \rightarrow e \prec e''$

Collecte des informations

- Un site collecteur récupère des informations sur le calcul, puis reconstruit.
- Traditionnellement : tout événement « significatif » est envoyé au site collecteur,
- ici : seuls les événements de fin de chemin sont transmis au collecteur.
- Traditionnellement, après collecte, le calcul est reconstruit puis rejoué,
- ici, on peut reconstruire et rejouer de manière entrelacée.

Représentation du type abstrait arbre

- Vision inductive :
arbre = Vide | Nœud (info, liste d'arbre)
- Vision graphe :
arbre = ensemble de chemins (propriété de préfixe)
- Vision vecteurs de chemin :
arbre = ensemble de feuilles annotées

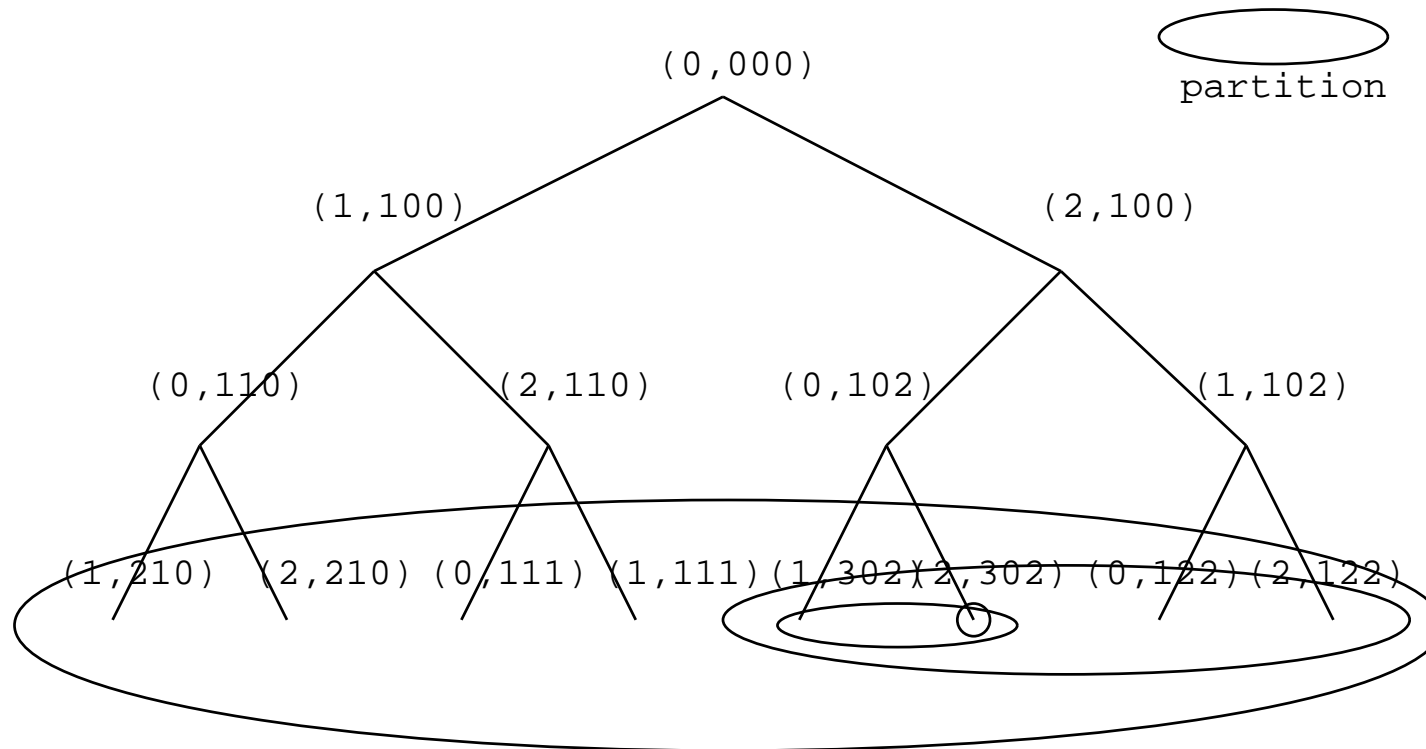
Interprétation de la migration

split = insérer(arbre, point d'insertion, nouvelles feuilles)

- (inductif) : identification du point d'insertion par la liste des nœuds de la racine au nœud feuille
- (graphe) :
 - point d'insertion identifié par un chemin
 - suppression du chemin
 - insertion de d nouveaux chemins construits à partir de l'ancien chemin et des nouvelles feuilles
- (vecteurs) :
 - point d'insertion identifié par un vecteur
 - suppression du vecteur
 - ajout de nouveaux vecteurs

Reconstruction

Nœuds feuilles \Rightarrow retrouver les nœuds internes.



Modélisation du protocole

```
const N : posnat
type Site = 0..N-1
  path_vector=Site × array Site of nat -- vecteur de chemin
Program basic_path_vectors
declare st: set of path_vector -- ensemble des vecteurs
always
  ...
initially st = { (0, (0, ...)) }
assign -- sur le site s, split atomique du vecteur (s, V) sur les sites de D
  ⟨ ∥ (s, V, D) : g(s, V, D) : st := st \ {(s, V)} ∪ nv(s, V, D) ⟩
end
```

Modélisation de la reconstruction

```
declare st: set of path_vector -- ensemble des vecteurs
        a: set of path -- arbre de calcul
        v2p: array st of path -- association vecteur-chemin
always
    ...
initially st = { (0, (0, ...)) }
assign -- sur le site s, split atomique du vecteur (s, V) sur D
    <|| (s, V, D): g(s, V, D) : st := st \ {(s, V)} ∪ nv(s, V, D)
        || a := a \ v2p[(s, V)] ∪ {v2p[(s, V)]; v | v ∈ nv(s, V, D)}
        || <|| v: v ∈ nv(s, V, D) : v2p[v] := v2p[(s, V)]; v>
    >
end
```

Reconstruction et Unicité

- **rec** : fonction de reconstruction.
 - **rec** : set of $\text{path_vector} \times \text{vector} \times \text{path_vector} \rightarrow \text{path}$
 - **rec** est une fonction *globale* de reconstruction du chemin d'une feuille identifiée par (s, V) :

$$\text{rec}(\underline{\text{st}}, v_init, (s, V))$$

- reconstruction et unicité:

$$\text{always_true } \mathfrak{S}(\text{rec}(\text{st}, v_init), \text{st}) = a$$

Reconstruction des chemins

1. Reconstruction des chemins sans localisation des vecteurs.

```
rec(st, init, (s, V)) = if st = {(s, V)} then [(s, V)]  
                      else let part = P(st)((s, V)) in  
                        cons(init, rec(part, min(part), (s, V)))  
                      end
```

2. Localisation des vecteurs.

Partitionnement

- Obtention des partitions = définition d'une relation d'équivalence
- deux vecteurs sont en relation s'ils ont une composante commune distincte de celle du vecteur minimale (ils sont dans le même sous-arbre)

$$\begin{aligned} R_1(\text{st})(i_1, i_2) = & \quad i_1 \in \text{st} \wedge i_2 \in \text{st} \\ & \wedge \quad i_1.V = \min(\text{st}) \vee i_2.V = \min(\text{st}) \Rightarrow i_1 = i_2 \\ & \wedge \quad i_1.V = i_2.V \vee \quad \exists s : s \neq i_1.w \wedge s \neq i_2.w \\ & \quad \wedge \quad i_1.V(s) = i_2.V(s) \\ & \quad \wedge \quad i_1.V(s) \neq \min(\text{st})(s) \end{aligned}$$

$$P(\text{st})(i) = [i]_{R_1^*}$$

Vecteurs de chemin et vecteurs d'horloges

- Les vecteurs de chemin ne s'appliquent qu'aux calculs diffusants.
- Les vecteurs de chemin (VC) et les vecteurs d'horloges (VH) enregistrent des événements différents :
 - VC : événement = split (réception-et-diffusion)
 - VH : événement = émission ou réception ou événement interne
- Les vecteurs d'horloges acquièrent une connaissance globale d'autres vecteurs en les "croisant" sur un site.
- Les vecteurs de chemin sont creux : seuls les sites visités induisent une composante non nulle.

Conclusion

- Reconstruction d'un calcul diffusant à partir des événements terminaux.
- Raisonnement sur un arbre.
- Encodage original d'un arbre.

Travaux en cours:

- Validation du développement.

Perspectives :

- clichés (reconstruction à la volée).
- extension du modèle : fusion de chemins.