K-parmi-L exclusion auto-stabilisante

Ajoy K. Datta University of Nevada
Stéphane Devismes CNRS, LRI
Florian Horn LIAFA
Lawrence L. Larmore University of Nevada
Présenté par
Samuel Bernard UPMC/LIP6

Algotel'2008, Saint-Malo

13 mai 2008

K-parmi-L exclusion [Raynal, 1991]

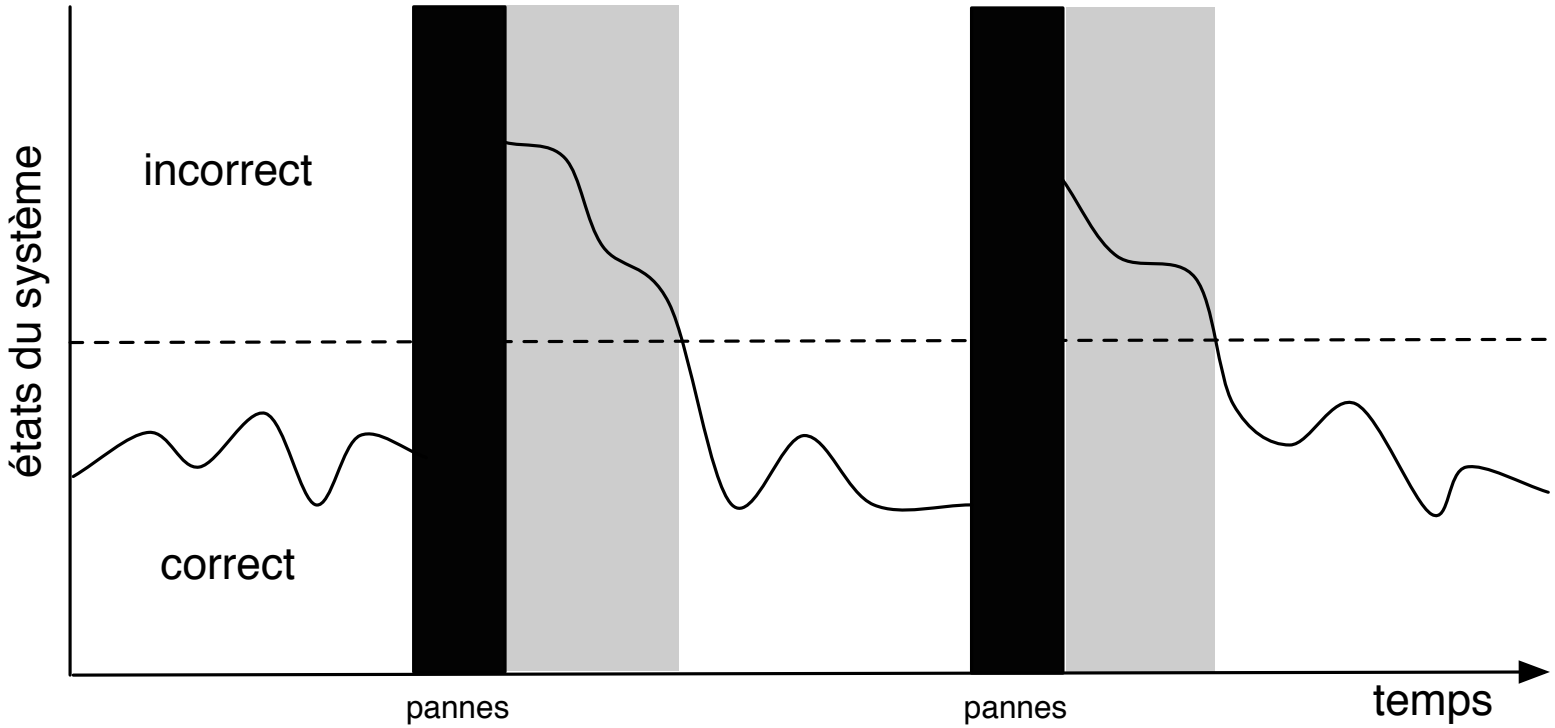
Problème de partage de ressources

- L unités d'une même ressource
- Chaque processeur peut demander jusqu'à K unités ($K \leq L$)

Spécification : *sûreté*, *vivacité* et *efficacité*

- **sûreté :** Chaque unité de la ressource partagée peut être utilisée par au plus un processeur à la fois et au plus L unités peuvent être utilisées simultanément
- **vivacité :** Toute demande d'au plus K unités doit être satisfaite en un temps fini
- **efficacité :** Le plus grand nombre possible de demandes doit être satisfait simultanément

Auto-stabilisation [Dijkstra, 1974]



Tolérance aux pannes

Technique générale pour tolérer les fautes transitoires

Travaux précédents

Deux approches classiques

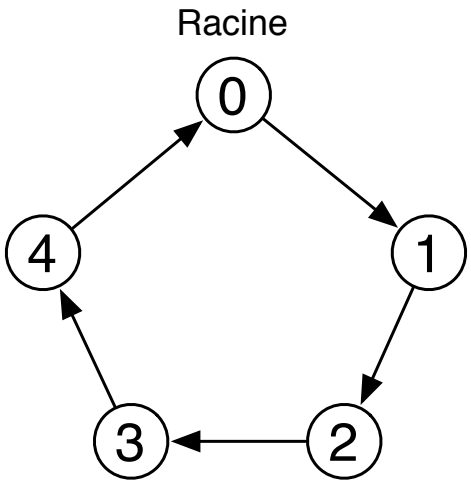
- *Protocoles à permissions* [Raynal, 1991] (solution non auto-stabilisante)
- *Circulation de jetons* [Datta, Hadid et Villain, 2003] (solution auto-stabilisante, temps de stabilisation : $5n$ rondes)

Plan

Modèle et approche

Modèle

- Passage de messages à capacité borné ($\mathcal{C}_{\max}$) et FIFO
- Anneau unidirectionnel et enraciné

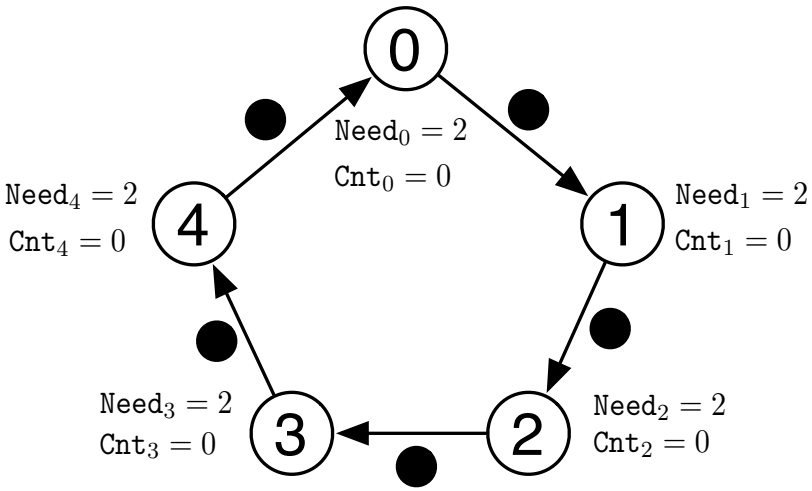


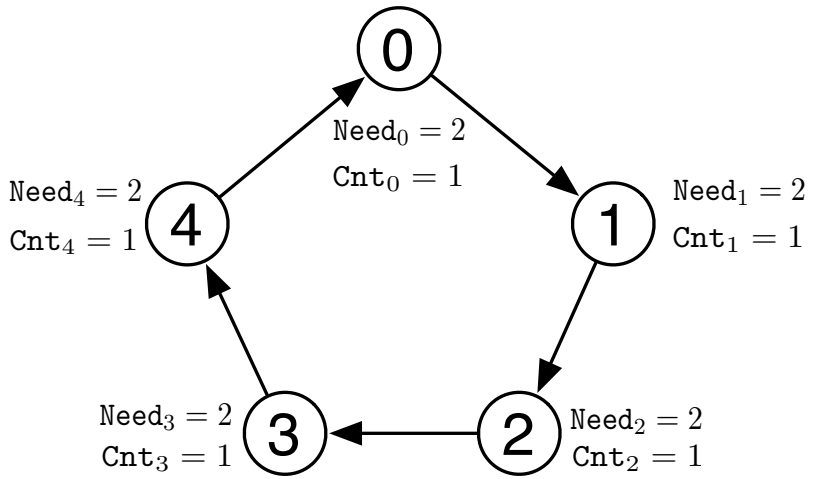
Approche modulaire

- Protocole naïf non-auto-stabilisant
- Régulateur pour l'auto-stabilisation

Ressources = Jetons

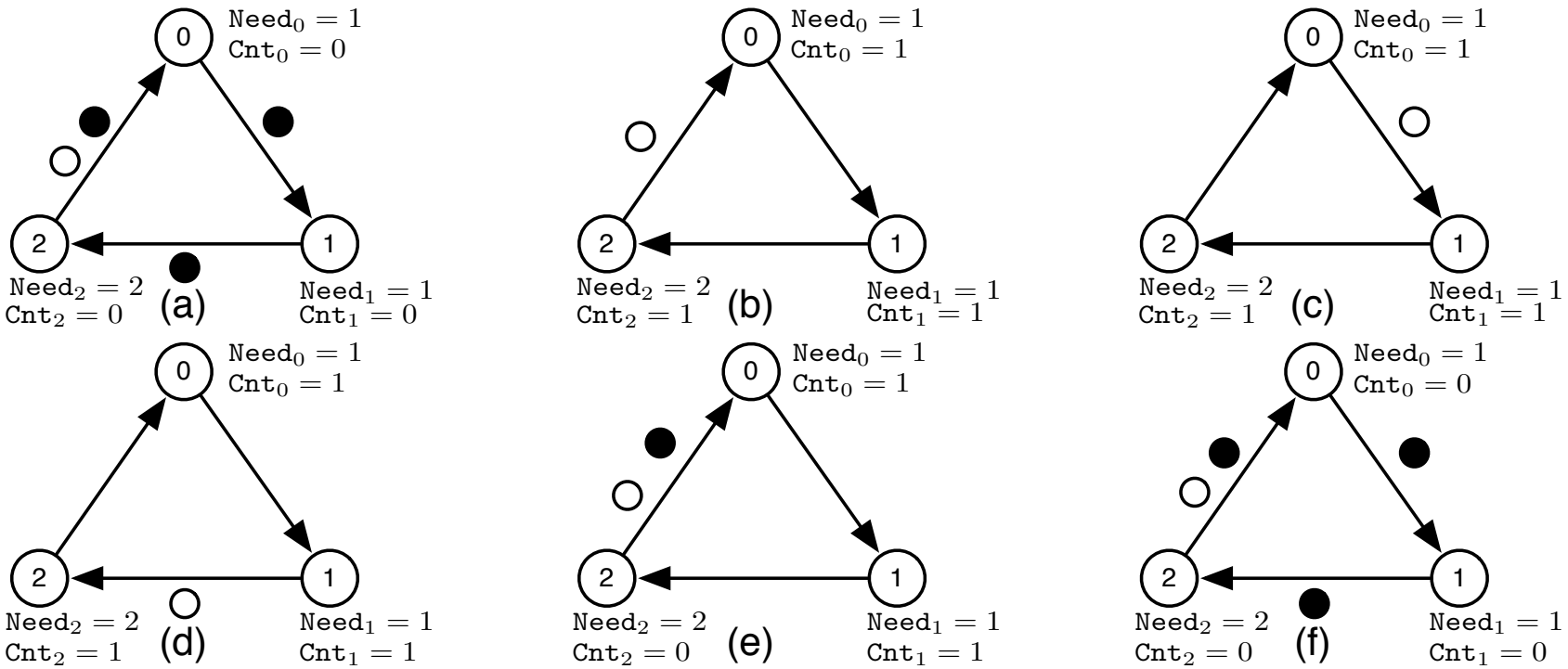
- Circulation de $\mathtt{L}$ jetons ressources
- Problème d'interblocage :





Jeton pousseur

- Sur réception du jeton pousseur, un processeur « relâche » ses jetons ressources
- Problème de famine :



Jeton priorité

- Annule l'effet du jeton pousseur
- Le jeton priorité est conservé par le processeur jusqu'à satisfaction de sa demande
- Un processeur qui possède le jeton priorité n'est pas tenu de relâcher ses jetons s'il reçoit le jeton pousseur : il renvoie uniquement le jeton pousseur à son successeur

Compter les jetons avec un jeton privilège

Le *jeton privilège* sert de point de repère

Entre deux réceptions du privilège, le processeur racine compte le nombre de jetons de chaque type et ajuste ce nombre si celui-ci n'est pas adéquat

Attention : le *jeton privilège* peut doubler des jetons

- Lorsqu'un processeur a réservé des jetons ressource et qu'il reçoit le jeton privilège, il garde ses jetons ressource mais renvoie le jeton privilège (pour éviter les interblocages)
- Pour ne pas oublier ces jetons, le processeur met à jour le champ « nombre de jetons doublés » du jeton privilège avant de le renvoyer

Stabiliser le jeton privilège [Varghese, 2000]

- Une variable C_i variant de 0 à à $n(\mathcal{C}_{\max}+1)-1$
- Chaque processeur envoie *périodiquement* à son successeur des messages avec cette valeur.
- Sur réception d'un message avec la valeur c :
  - La racine considère qu'un message est un jeton privilège valide si $c = C_0$. Dans ce cas, elle incrémente C_0 modulo $n(\mathcal{C}_{\max}+1)$.
  - Le processeur $i \neq 0$ considère qu'un message est un jeton privilège valide si $c \neq C_i$. Dans ce cas, il donne à C_i la valeur c.

Conclusion

- Approche intuive mais efficace : temps de stabilisation meilleur que pour les autres solutions ($4n$ rondes)
- Extension possible au réseau enraciné quelconque (en utilisant une circulation de jeton en profondeur comme [Petit et Villain, 1997])
- Approche modulaire appliquable à d'autres problèmes