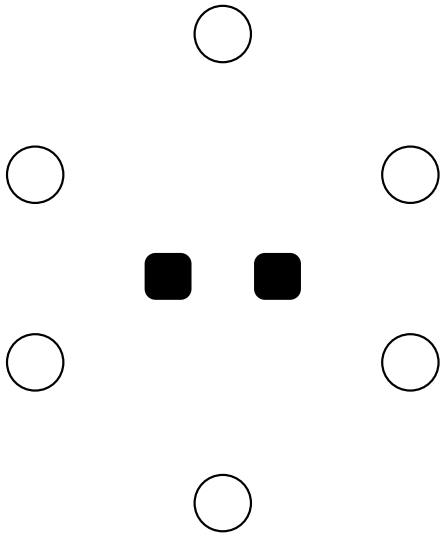
Allocation de ressources tolérante aux fautes utilisant des détecteurs de défaillances

Mathieu Bouillaguet
Pierre Sens Luciana Arantes

Université Pierre & Marie Curie, Paris VI

Laboratoire d'informatique de Paris VI

Section critique à entrées multiples

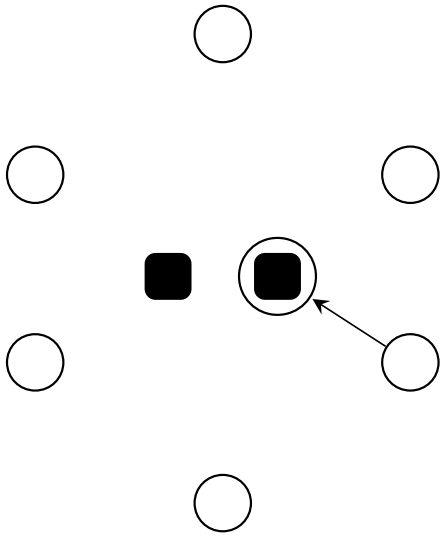


- n nœuds
- k ressources
- 1 seule ressource allouée à la fois

Propriétés à vérifier

- Sûreté
- Vivacité

Section critique à entrées multiples

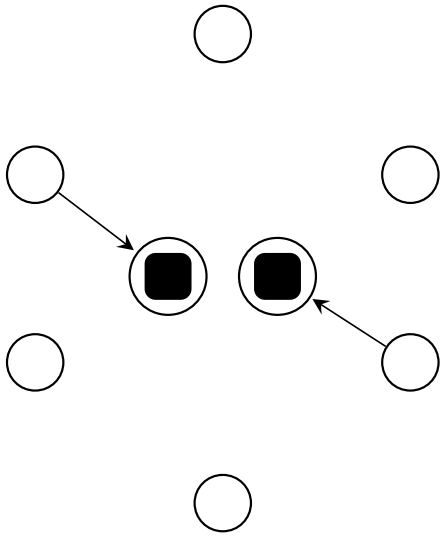


- n nœuds
- k ressources
- 1 seule ressource allouée à la fois

Propriétés à vérifier

- Sûreté
- Vivacité

Section critique à entrées multiples

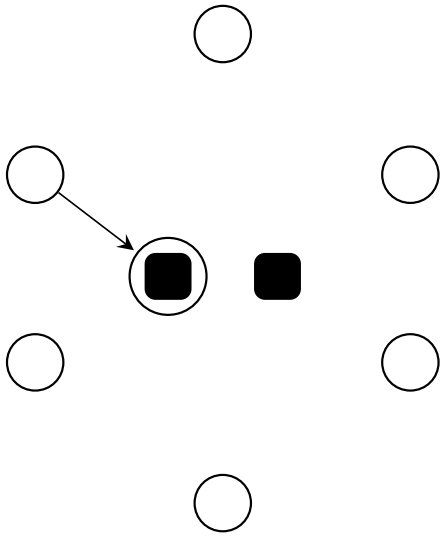


- n nœuds
- k ressources
- 1 seule ressource allouée à la fois

Propriétés à vérifier

- Sûreté
- Vivacité

Section critique à entrées multiples

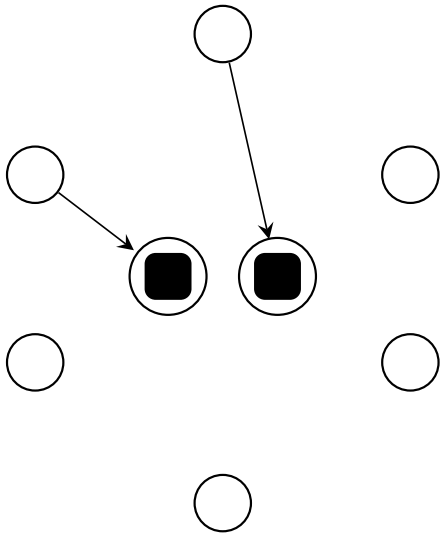


- n nœuds
- k ressources
- 1 seule ressource allouée à la fois

Propriétés à vérifier

- Sûreté
- Vivacité

Section critique à entrées multiples

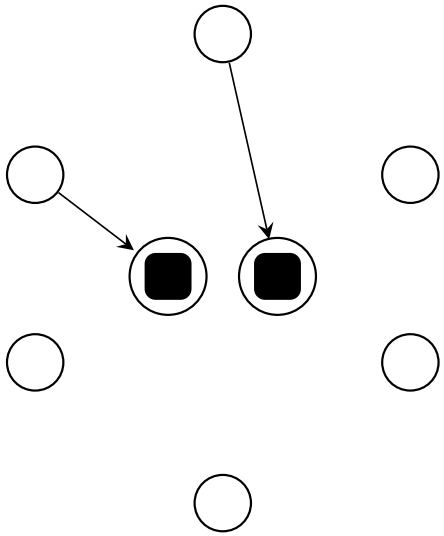


- n nœuds
- k ressources
- 1 seule ressource allouée à la fois

Propriétés à vérifier

- Sûreté
- Vivacité

Section critique à entrées multiples



- n nœuds
- k ressources
- 1 seule ressource allouée à la fois

Propriétés à vérifier

- Sûreté
- Vivacité

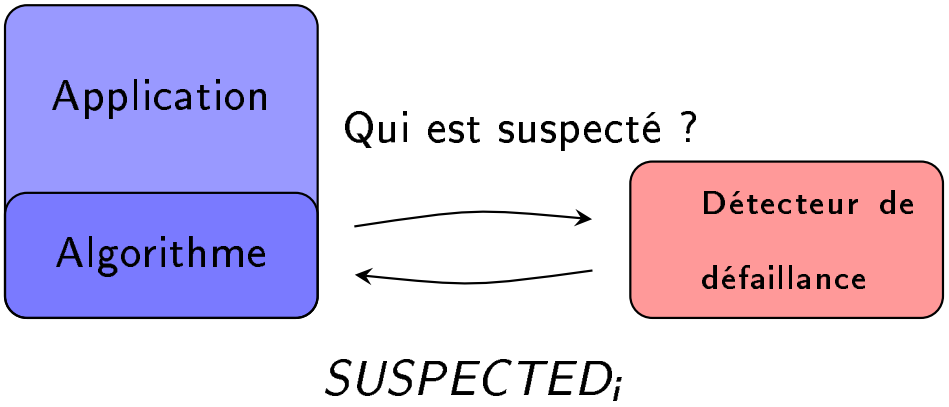
Modèle

- Communication par envoi de messages
- Système asynchrone
  pas de bornes sur :
  - les délais d'acheminement des messages
  - la vitesse relative des processus
- Canaux fiables
- Défaillances de type pannes franches

Détecteur de défaillance (Chandra et Toueg)

Definition

Un oracle distribué qui fournit une liste de sites suspectés d'avoir crashé.



Avantages

- Algorithmes plus simples
- Solutions génériques

Classes de détecteur de défaillance

Complétude

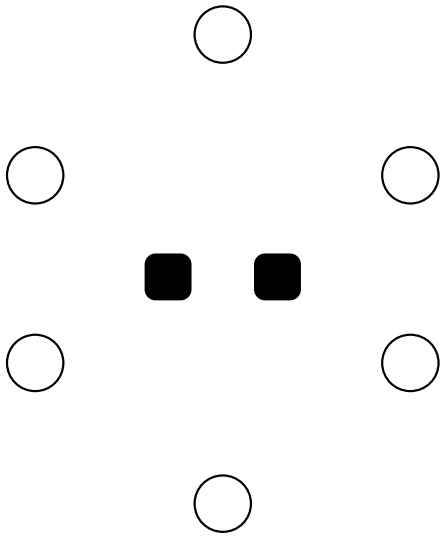
Spécifie comment les processus crashés sont détectés

Justesse

Limite les erreurs que peut faire un détecteur

		Justesse			
		Forte	Faible	Finalement Forte	Finalement Faible
Complétude	Forte	Parfait $\mathcal{P}$	Fort $\mathcal{S}$	Finalement Parfait $\diamond\mathcal{P}$	Finalement Fort $\diamond\mathcal{S}$
	Faible	$\mathcal{Q}$	Faible $\mathcal{W}$	$\diamond\mathcal{Q}$	Finalement Faible $\diamond\mathcal{W}$

Algorithme de Raymond

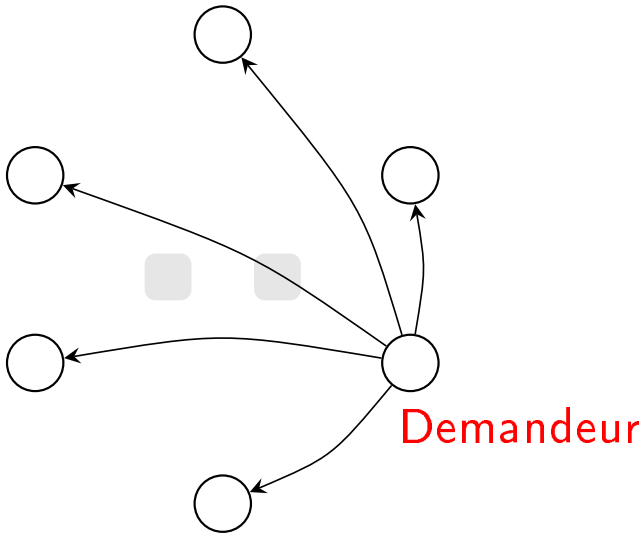


- $n = 6$ nœds
- $k = 2$ ressources

$n - k$ permissions sont nécessaires

Cela garantie qu'une ressource est disponible

Algorithme de Raymond

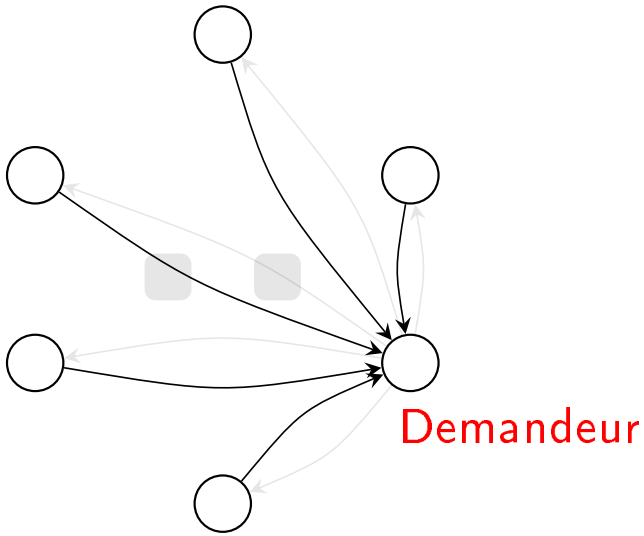


- $n = 6$ nœds
- $k = 2$ ressources

$n - k$ permissions sont nécessaires

Cela garantie qu'une ressource est disponible

Algorithme de Raymond

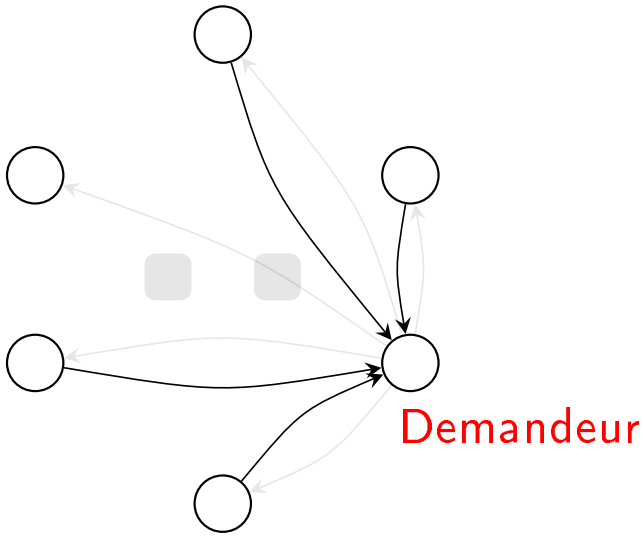


- $n = 6$ nœds
- $k = 2$ ressources

$n - k$ permissions sont nécessaires

Cela garantie qu'une ressource est disponible

Algorithme de Raymond

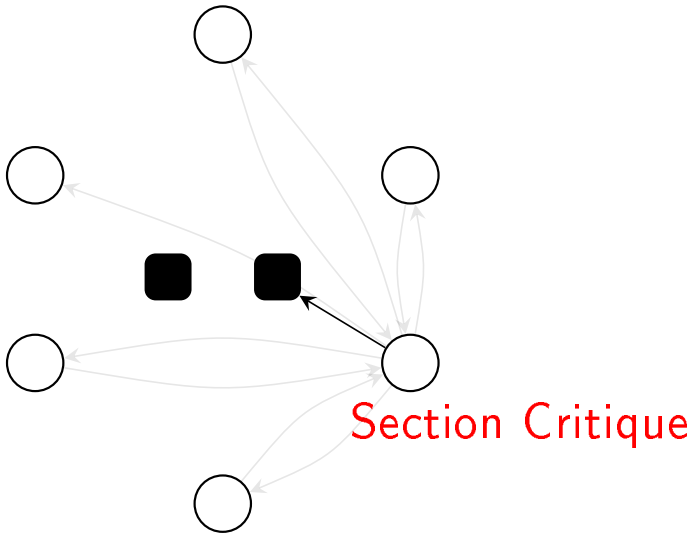


- $n = 6$ nœds
- $k = 2$ ressources

$n - k$ permissions sont nécessaires

Cela garantie qu'une ressource est disponible

Algorithme de Raymond



- $n = 6$ nœds
- $k = 2$ ressources

$n - k$ permissions sont nécessaires

Cela garantie qu'une ressource est disponible

Le détecteur $\mathcal{T}$

Propriétés

- Complétude forte
- Justesse finalement forte
- Justesse de confiance

$\mathcal{T}$ est plus faible que $\mathcal{P}$

$\diamond\mathcal{P} \preceq \mathcal{T} \preceq \mathcal{P}$

$\mathcal{T}$ ne fait plus de fausses suspicions sur un processus p_i s'il a fait confiance à p_i

Le détecteur $\mathcal{T}$

Propriétés

- Complétude forte
- Justesse finalement forte
- Justesse de confiance

$\mathcal{T}$ est plus faible que $\mathcal{P}$

$\diamond\mathcal{P} \preceq \mathcal{T} \preceq \mathcal{P}$

$\mathcal{T}$ ne fait plus de fausses suspicions sur un processus p_i s'il a fait confiance à p_i

Algorithme tolérant aux fautes

1. Phase d'initialisation (permet d'avoir la confiance d'au moins un processus).
2. Chaque processus a la confiance d'au moins un processus *correct*
3. Un détecteur $\mathcal{S}$ est nécessaire pour l'initialisation
4. A la réception d'un message *CRASH* la vision locale des sites vivants est mise à jour (n dynamique)

Algorithme tolérant aux fautes

1. Phase d'initialisation (permet d'avoir la confiance d'au moins un processus).
2. Chaque processus a la confiance d'au moins un processus *correct*
3. Un détecteur $\mathcal{S}$ est nécessaire pour l'initialisation
4. A la réception d'un message *CRASH* la vision locale des sites vivants est mise à jour (n dynamique)

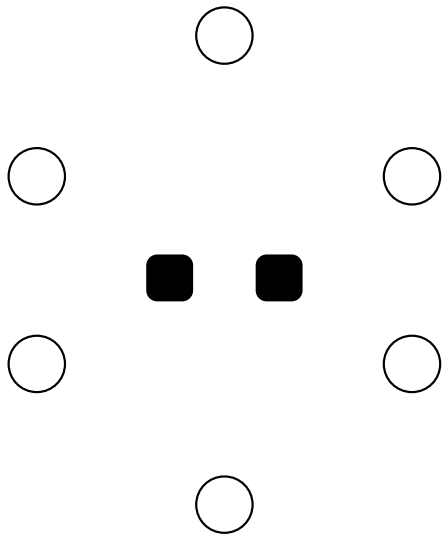
Algorithme tolérant aux fautes

1. Phase d'initialisation (permet d'avoir la confiance d'au moins un processus).
2. Chaque processus a la confiance d'au moins un processus *correct*
3. Un détecteur $\mathcal{S}$ est nécessaire pour l'initialisation
4. A la réception d'un message *CRASH* la vision locale des sites vivants est mise à jour (n dynamique)

Algorithme tolérant aux fautes

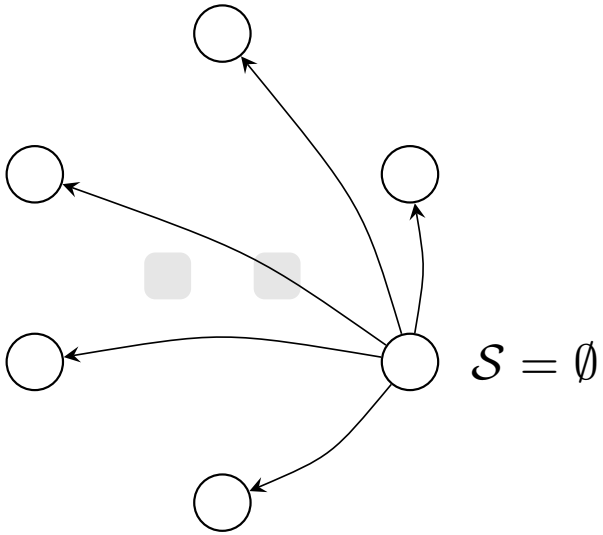
1. Phase d'initialisation (permet d'avoir la confiance d'au moins un processus).
2. Chaque processus a la confiance d'au moins un processus *correct*
3. Un détecteur $\mathcal{S}$ est nécessaire pour l'initialisation
4. A la réception d'un message *CRASH* la vision locale des sites vivants est mise à jour (n dynamique)

Phase d'initialisation



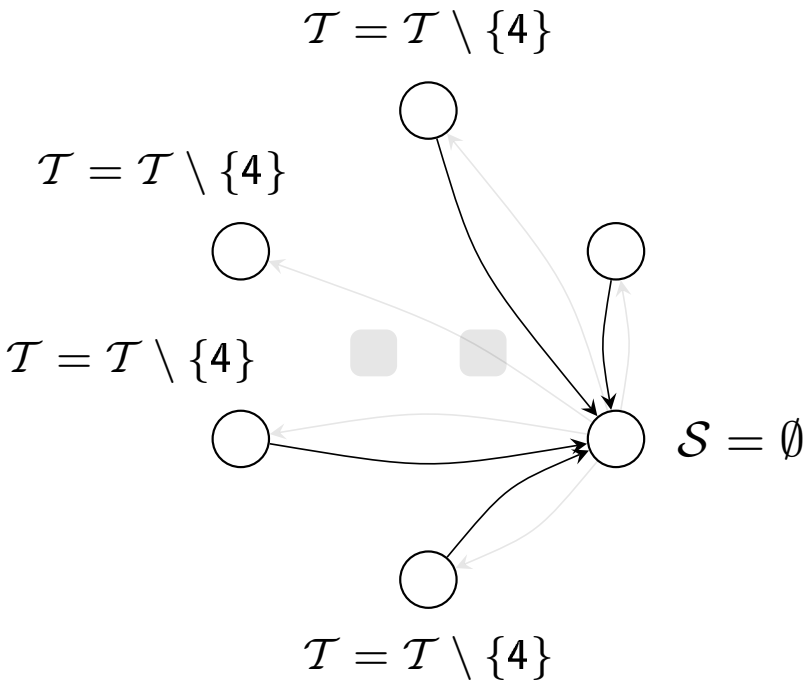
Upon receive $ME(m)$ do
wait until $m \notin \mathcal{T}$
$trusted_i := trusted_i \cup \{m\}$
send ACK to m

Phase d'initialisation



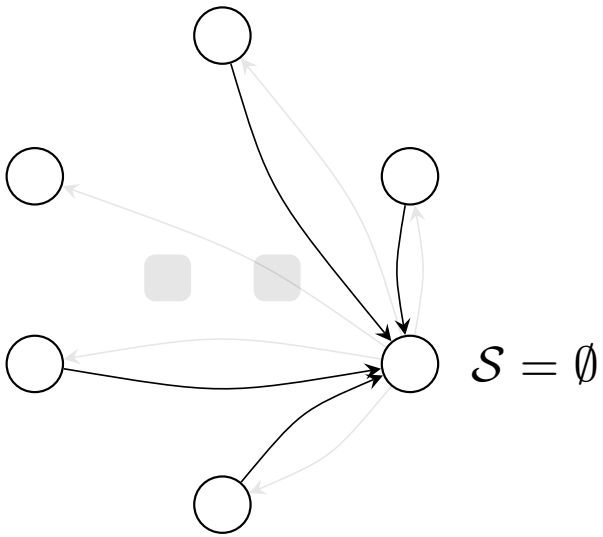
Upon receive $ME(m)$ do
wait until $m \notin \mathcal{T}$
$trusted_i := trusted_i \cup \{m\}$
send ACK to m

Phase d'initialisation



Upon receive $ME(m)$ do
wait until $m \notin \mathcal{T}$
$trusted_i := trusted_i \cup \{m\}$
send ACK to m

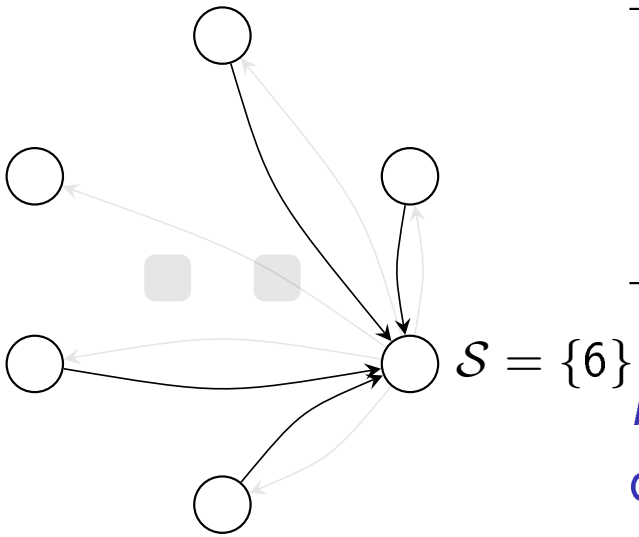
Phase d'initialisation



Upon receive $ME(m)$ do
wait until $m \notin \mathcal{T}$
$trusted_i := trusted_i \cup \{m\}$
send ACK to m

m attend un acquittement de tous les processus qui ne sont pas suspectés par $\mathcal{S}$

Phase d'initialisation



Upon receive $ME(m)$ do
wait until $m \notin \mathcal{T}$
$trusted_i := trusted_i \cup \{m\}$
send ACK to m

m attend un acquittement de tous les processus qui ne sont pas suspectés par $\mathcal{S}$

Détection de crash

upon $(l \in \mathcal{T})$ and $(l \in trusted_i)$
$trusted_i := trusted_i \setminus \{l\}$
send $(CRASH)$ to all

upon receive $CRASH(l)$ do
if $l \notin crashed_i$ then
$n - -$
$crashed_i := crashed_i \cup \{l\}$

Clause d'entrée en section critique

$perm\_count - i \geq n - k$

Détection de crash

upon $(l \in \mathcal{T})$ and $(l \in trusted_i)$
$trusted_i := trusted_i \setminus \{l\}$
send $(CRASH)$ to all

upon receive $CRASH(l)$ do
if $l \notin crashed_i$ then
$n - -$
$crashed_i := crashed_i \cup \{l\}$

Clause d'entrée en section critique

$perm\_count - i \geq n - k$

Détection de crash

upon $(l \in \mathcal{T})$ and $(l \in trusted_i)$
$trusted_i := trusted_i \setminus \{l\}$
send $(CRASH)$ to all

upon receive $CRASH(l)$ do
if $l \notin crashed_i$ then
$n - -$
$crashed_i := crashed_i \cup \{l\}$

Clause d'entrée en section critique

$perm\_count - i \geq n - k$

Limitations

En cas de crash avant l'initialisation, il n'y a pas de moyen de détecter la panne!

Au delà de $k-1$ crashs initiaux l'algorithme est bloqué

Limitations

En cas de crash avant l'initialisation, il n'y a pas de moyen de détecter la panne!

Au delà de $k-1$ crashs initiaux l'algorithme est bloqué

Conclusion

En cours

- En cours de rédaction
- Mesure des performances (simulation)

Perspectives

- Extension à d'autres problèmes
- Identifier un détecteur minimal pour la section critique à entrées multiples
- Adapter à une topologie particulière (grilles, ...)


Tushar Deepak Chandra and Sam Toueg.
Unreliable failure detectors for reliable distributed systems.
Journal of the Association for Computing Machinery, 43(2):225–267, March 1996.


Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Petr Kouznetsov.
Mutual exclusion in asynchronous systems with failure detectors.
J. Parallel Distrib. Comput., 65(4):492–505, 2005.


A. Mostefaoui, E. Mourgaya, and M. Raynal.
Asynchronous implementation of failure detectors.
Dependable Systems and Networks, 2003. Proceedings. 2003 International Conference on, pages 351–360, 2003.


Kerry Raymond.
A distributed algorithm for multiple entries to a critical section.
Inf. Process. Lett., 30(4):189–193, 1989.