Comment se mettre d'accord quand les autres dorment ?

Anaïs Durand, Michel Raynal, Gadi Taubenfeld

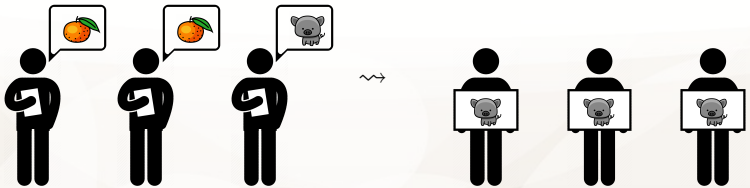


26 Mai 2023

UNIVERSITÉ Clermont Auvergne

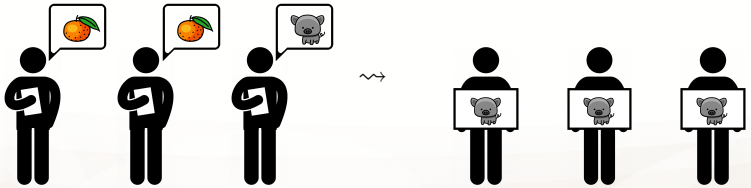
Consensus

Objet partagé, one-shot, avec opération `propose`



Consensus

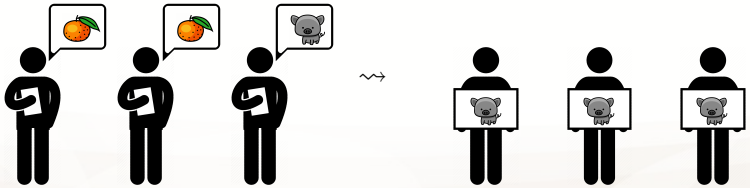
Objet partagé, one-shot, avec opération `propose`



- Validité :

Consensus

Objet partagé, one-shot, avec opération `propose`

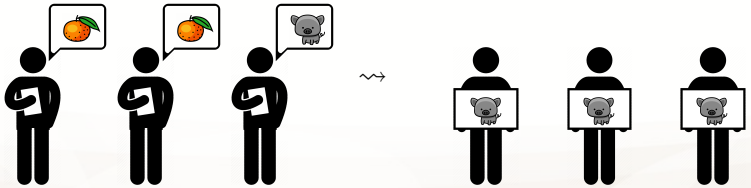


- Validité :

- Accord :

Consensus

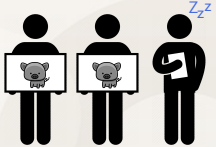
Objet partagé, one-shot, avec opération `propose`



- Validité :

- Accord :

- Terminaison :



UNIVERSITÉ Clermont Auvergne

État de l'art dans les systèmes asynchrones

- **Impossible** dans les systèmes asynchrones à **passage de messages**
  [*Fischer, Lynch, Paterson, 1985*]

- **Impossible** dans les systèmes asynchrones en **lecture/écriture**
  [*Loui, Abu-Amara, 1987*]

État de l'art dans les systèmes asynchrones

- **Impossible** dans les systèmes asynchrones à **passage de messages**
  [*Fischer, Lynch, Paterson, 1985*]

- **Impossible** dans les systèmes asynchrones en **lecture/écriture**
  [*Loui, Abu-Amara, 1987*]

- **Possible** dans les systèmes asynchrones à **passage de messages** si :
  - une **majorité** de processus ne tombe pas en panne
  - les pannes sont **initiales**

  [*Fischer, Lynch, Paterson, 1985*]

Est-il **possible** de résoudre

le **consensus asynchrone**

si les pannes ont lieu

avant un **seuil de contention** prédéfini ?

Modèle de calcul

- n processus $p_1, p_2, ..., p_n$
- Registres atomiques en lecture/écriture
- Participation des processus

non participation $\equiv$ panne initiale

- **Contention :** # processus ayant accédé à un registre partagé

[*Taubenfeld, 2018*], [*Durand, Raynal, Taubenfeld, 2022*]

Modèle de calcul

- n processus $p_1, p_2, ..., p_n$
- Registres atomiques en lecture/écriture
- Participation des processus

non participation $\equiv$ panne initiale

- **Contention :** # processus ayant accédé à un registre partagé

[*Taubenfeld, 2018*], [*Durand, Raynal, Taubenfeld, 2022*]

UNIVERSITÉ Clermont Auvergne

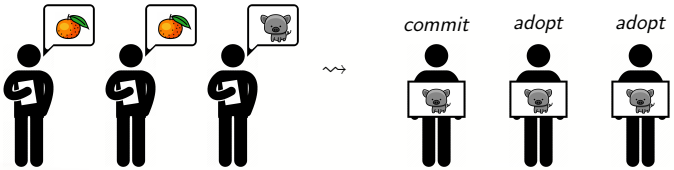
Consensus asynchrone avec :

- 1 panne λ-contrainte
- $\lambda = n - 1$

Adopt-commit

[*Gafni, 1998*]

Objet partagé, one-shot, avec opération `ac_propose`

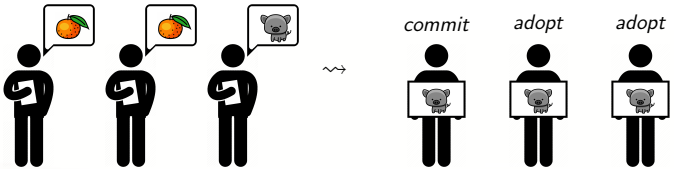


Terminaison + **validité** du consensus

Adopt-commit

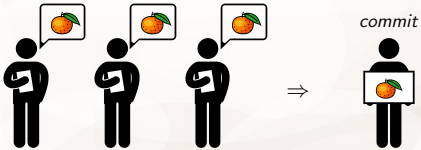
[*Gafni, 1998*]

Objet partagé, one-shot, avec opération `ac_propose`



Terminaison + **validité** du consensus

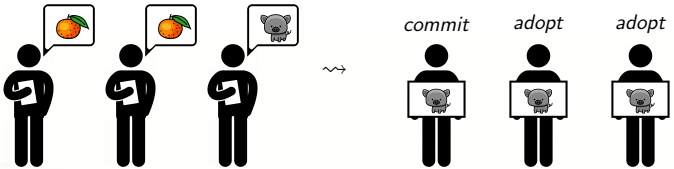
- **Obligation :**



Adopt-commit

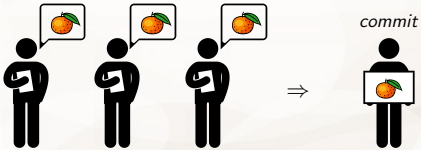
[*Gafni, 1998*]

Objet partagé, one-shot, avec opération `ac_propose`

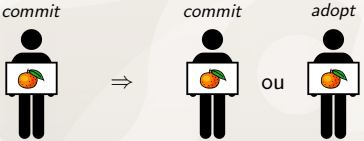


Terminaison + **validité** du consensus

- **Obligation :**



- **Accord faible :**



UNIVERSITÉ Clermont Auvergne

Algorithme ($k = 1$, $\lambda = n - 1$)

Objets partagés :

- $INPUT[1..n]$: tableau de registres atomiques, initialisés à $\perp$
- DEC : registre atomique, initialisé à $\perp$
- $LAST$: registre atomique
- AC : objet adopt/commit

Algorithme ($k = 1,\ \lambda = n - 1$)

propose(in_i) {
    $INPUT[i] := in_i$;

}

Algorithme ($k = 1$, $\lambda = n - 1$)

propose(in_i) {

$INPUT[i] := in_i$;

repeat {

$input1_i :=$ lecture (non-atomique) de $INPUT[1..n]$;

$input2_i :=$ lecture (non-atomique) de $INPUT[1..n]$;

} **until** ($input1_i = input2_i \wedge input1_i$ contient au plus un $\perp$);

}

Algorithme ($k = 1$, $\lambda = n - 1$)

propose(in_i) {

$INPUT[i] := in_i$;

repeat {

$input1_i :=$ lecture (non-atomique) de $INPUT[1..n]$;

$input2_i :=$ lecture (non-atomique) de $INPUT[1..n]$;

} **until** ($input1_i = input2_i \wedge input1_i$ contient au plus un $\perp$);

$val_i :=$ **min**(valeurs dans $input1_i[1..n]$);

if ($\exists j : input1_i[j] = \perp$) { $LAST := j$; }

}

Algorithme ($k = 1$, $\lambda = n - 1$)

propose(in_i) {

$INPUT[i] := in_i$;

repeat {

$input1_i :=$ lecture (non-atomique) de $INPUT[1..n]$;

$input2_i :=$ lecture (non-atomique) de $INPUT[1..n]$;

} **until** ($input1_i = input2_i \wedge input1_i$ contient au plus un $\perp$);

$val_i :=$ **min**(valeurs dans $input1_i[1..n]$);

if ($\exists j : input1_i[j] = \perp$) { $LAST := j$; }

$\langle tag_i, res_i \rangle := AC.ac\_propose(val_i)$;

if ($tag_i = commit \vee LAST = i$) {

$DEC := res_i$;

} **else** {

attendre jusqu'à ce que $DEC \neq \perp$;

}

return DEC;

}

Consensus asynchrone avec :

- $k \geq 1$ pannes λ-contrainte
- $\lambda = n - k$
- $+$ des objets de nombre-consensus k

Nombre-consensus

[*Herlihy, 91*]

$NC(O)$ = **Nombre maximum** de processus tel que le **consensus** est **possible** :

- quel que soit le nombre de pannes
- avec autant d'objets O et de registres L/E que nécessaire

Exemples :

- NC(Registres atomiques L/E) $= 1$
- NC(Liste FIFO) $= 2$
- NC(Compare & swap) $= \infty$

Algorithme ($k \geq 1$, $\lambda = n - k$)

Objets partagés :

- $KCONS[1..\lceil n/k \rceil]$: tableau d'objets *kCONS*
- $PARTICIPANT[1..n]$: tableau de booléens, initialisés à `false`

Algorithme ($k \geq 1$, $\lambda = n - k$)

Objets partagés :

- $KCONS[1..\lceil n/k \rceil]$: tableau d'objets *kCONS*
- $PARTICIPANT[1..n]$: tableau de booléens, initialisés à `false`

Idée de l'algorithme :

- **Clusters** de processus partageants le même objet *kCONS* $p_i, p_j, \ldots$ tels que $\lceil i/k \rceil = \lceil j/k \rceil$
- Un cluster **simule** un unique processus du 1er algorithme

Algorithme ($k \geq 1$, $\lambda = n - k$)

propose(in_i) {
    $PARTICIPANT[i]$:= `true`;
    repeat {
        $participant_i$:= lecture (non-atomique) de $PARTICIPANT[i]$;
    } **until** ($participant_i$ contient au plus k `false`);

Algorithme ($k \geq 1$, $\lambda = n - k$)

propose(in_i) {
  $PARTICIPANT[i]$:= `true`;
  repeat {
    $participant_i$:= lecture (non-atomique) de $PARTICIPANT[i]$;
  } **until** ($participant_i$ contient au plus k `false`);
  $INPUT[\lceil i/k \rceil] := KCONS[\lceil i/k \rceil].propose(in_i)$;
  repeat {
    $input1_i$:= lecture (non-atomique) de $INPUT[1..\lceil n/k \rceil]$;
    $input2_i$:= lecture (non-atomique) de $INPUT[1..\lceil n/k \rceil]$;
  } **until** ($input1_i = input2_i \wedge input1_i$ contient au plus un $\perp$);

Algorithme ($k \geq 1$, $\lambda = n - k$)

val_i := **min**(valeurs dans $input1_i[1..\lceil n/k \rceil]$);

if $(\exists j : input1_i[j] = \bot)$ { $LAST := j$; }

}

Algorithme ($k \geq 1$, $\lambda = n - k$)

val_i := **min**(valeurs dans $input1_i[1..\lceil n/k \rceil]$);

if $(\exists j : input1_i[j] = \bot)$ { $LAST := j$; }

$\langle tag_i, res_i \rangle := AC.ac\_propose(val_i)$;

if $(tag_i = commit \vee LAST = \lceil i/k \rceil)$ {

    $DEC := res_i$;

} **else** {

    attendre jusqu'à ce que $DEC \neq \bot$;

}

return DEC;

}

Conclusion

- Étude du **pouvoir** et des **limites** des pannes liées à la contention
- Extension des modèles dans lesquels le **consensus** est **possible**

λ	pannes	possible ?	
$n-1$	1	✓	
	≥ 2	✗	
$n-k$	$\leq k$	✓	(avec objets de nombre-consensus k)
	$> k$	✗	(même avec objets de nombre-consensus k)
$n-2k+1$	$\leq n-2k-1$	✓	(avec objets de nombre-consensus k et n divisible par k)