Proof-of-Stake

Introduction au problème du consensus distribué

Sylvain Conchon

Laboratoire Méthodes Formelles

Université Paris-Saclay, CNRS, ENS Paris-Saclay

sylvain.conchon@universite-paris-saclay.fr

Synchronisme ?

Rappel :

(H3) Communications synchrones : les nœuds partagent une horloge globale et tous les messages envoyés au temps t arrivent à $t + \Delta$, avec Δ une constante fixe et connue de tous.

Synchronisme ?

Rappel :

(H3) Communications synchrones : les nœuds partagent une horloge globale et tous les messages envoyés au temps t arrivent à $t + \Delta$, avec Δ une constante fixe et connue de tous.

Est-ce que cette hypothèse est réaliste dans le domaine des blockchains ?

Peut-on la relâcher ?

Résultat d'impossibilité sous asynchronisme (FLP)

(H3') Communications asynchrones : aucune horloge globale, aucune garantie sur la délivrance des messages (pas de temps max, pertes de messages possibles, etc.).

Le résultat d'impossibilité est dû à Fischer, Lynch et Paterson (FLP85) : aucun algorithme déterministe de consensus π n'est possible dans un environnement asynchrone.

La preuve montre que si π existe et satisfait les propriétés Agreement et Validity, alors π doit nécessairement avoir une trace infinie et il ne respecte donc pas la propriété de terminaison.

Entre synchronisme et asynchronisme

On peut supposer un modèle de communication intermédiaire, le synchronisme partiel (*partial synchrony*), dans lequel :

- les nœuds partagent une horloge globale (hypothèse qui peut être assouplie en bornant les décalages d'horloges)
- on suppose que les communications sont asynchrones jusqu'à un temps GST (Global Stabilization Time), inconnu, puis synchone après GST.

Le réseau est donc asynchrone jusqu'à un certain temps (c'est la phase d'attaque), puis il devient synchrone (c'est la phase normale)

Ce mode de communication semble plus proche de la réalité (on suppose que lorsqu'un réseau ne fonctionne pas, il faut un peu de temps pour le réparer).

Résultat d'impossibilité du synchronisme partiel

Un résultat d'impossibilité de Dwork, Lynch et Stockmeyer de 1988, montre que :

> Le consensus distribué est impossible sous hypothèses de synchronisme partiel si $f \geq N/3$

Dépasser FLP

On peut dépasser le résultat d'impossibilité FLP dans les réseaux asynchones en concevant des algorithmes non-déterministes (en utilisant de l'aléatoire)

ALGORITHMES PROBABILISTES

Avalanche (1/2)

La blockchain Avalanche repose sur un algorithme de consensus basé sur un mécanisme de Métastabilité de type propagation épidémique.

Alors qu'un algorithme BFT traditionnel nécessite $O(n^2)$ messages pour qu'un ensemble de n nœuds converge, cet algorithme épidémique n'exige que $O(k \times n)$, avec $k \ll n$.

La sûreté d'Avalanche est probabiliste.

Avalanche (2/2)

On commence par présenter la méthode de métastabilité avec l'algorithme Slush. Puis on étend cet algorithme pour résister aux attaques Byzantines (Snowflake). Enfin, une dernière amélioration permet de renforcer encore l'algorithme (Snowball).

Avalanche (2/2)

On commence par présenter la méthode de métastabilité avec l'algorithme Slush. Puis on étend cet algorithme pour résister aux attaques Byzantines (Snowflake). Enfin, une dernière amélioration permet de renforcer encore l'algorithme (Snowball).

On suppose que la blockchain est constituée de $\mathcal{N}$ nœuds.

Métastabilité

L'algorithme Slush est basé sur une approche métastable inspirée d'un protocole épidémique. Les paramètres de l'algorithme sont m, k et α.

```
1:  procedure onQuery(v, col')
2:      if col = ⊥ then col := col'
3:      respond(v, col)
4:  procedure slushLoop(u, col_0 ∈ {R, B, ⊥})
5:      col := col_0 // initialize with a color
6:      for r ∈ {1 ... m} do
7:          // if ⊥, skip until onQuery sets the color
8:          if col = ⊥ then continue
9:          // randomly sample from the known nodes
10:         K := sample(N\u, k)
11:         P := [query(v, col)   for v ∈ K]
12:         for col' ∈ {R, B} do
13:             if P.count(col') ≥ α · k then
14:                 col := col'
15:     accept(col)
```

Slush : latence

$\mathcal{C}$ Size	600	1200	2400	4800	9600
Exp. Conv.	12.66	14.39	15.30	16.43	18.61

Table 1: Expected number of per-node-iterations to convergence starting at worst-case (equal) $\mathcal{C}$ network split, in the case of $k = 10$, $\alpha = 0.8$. Standard deviation for all samples is ≤ 2.5.

Snowflake

On renforce Slush en ajoutant un compteur cnt qui capture la conviction qu'à un nœud sur sa couleur. La valeur β est un paramètre de cet algorithme.

```
1:  procedure SNOWFLAKELOOP(u, col_0 ∈ {R, B, ⊥})
2:      col := col_0, cnt := 0
3:      while undecided do
4:          if col = ⊥ then continue
5:          K := SAMPLE(N \ u, k)
6:          P := [QUERY(v, col)   for v ∈ K]
7:          for col' ∈ {R, B} do
8:              if P.COUNT(col') ≥ α · k then
9:                  if col' ≠ col then
10:                     col := col', cnt := 0
11:                 else
12:                     if ++cnt > β then ACCEPT(col)
```

Snowball

On renforce encore l'algorithme en ajoutant une compteur de confiance d par couleur.

```
1:  procedure SNOWBALLLOOP(u, col_0 ∈ {R, B, ⊥})
2:      col := col_0, lastcol := col_0, cnt := 0
3:      d[R] := 0, d[B] := 0
4:      while undecided do
5:          if col = ⊥ then continue
6:          K := SAMPLE(N \ u, k)
7:          P := [QUERY(v, col)   for v ∈ K]
8:          for col' ∈ {R, B} do
9:              if P.COUNT(col') ≥ α · k then
10:                 d[col']++
11:                 if d[col'] > d[col] then
12:                     col := col'
13:                 if col' ≠ lastcol then
14:                     lastcol := col', cnt := 0
15:                 else
16:                     if ++cnt > β then ACCEPT(col)
```

Exercice 2

Réaliser un simulateur pour l'algorithme **Snowball**.

Ce programme pourra être développé d'une manière séquentielle, en utilisant par exemple un simple tableau pour représenter le maillage des noeuds du réseau, et un générateur de nombres aléatoires pour simuler les communications entre les noeuds.

Les pannes prises en compte par votre simulateur devront aller de simples pannes sèches (sans reprise) aux comportements Bizantins.