Blockchains : fondements et applications

Sylvain Conchon

Laboratoire Méthodes Formelles

Université Paris-Saclay, CNRS, ENS Paris-Saclay

sylvain.conchon@universite-paris-saclay.fr

Contenu du cours

- Rappels sur le problème SMR et l'architecture Primary-Backup
- Lock-Commit
- Tenderbake
- Consensus et comités

Rappel : le problème SMR

Le problème de consensus le plus proche de celui lié aux blockchains est le State Machine Replication (SMR) problem.

Il s'agit de permettre à des client d'accéder, en lecture et écriture, à une base de données. Afin de garantir un service sans faille et efficace, on décide de répliquer la base de données sur plusieurs machines. Cela pose le problème de garder tous les ordinateurs synchronisés.

Rappel : le problème SMR

Le problème de consensus le plus proche de celui lié aux blockchains est le State Machine Replication (SMR) problem.

Il s'agit de permettre à des client d'accéder, en lecture et écriture, à une base de données. Afin de garantir un service sans faille et efficace, on décide de répliquer la base de données sur plusieurs machines. Cela pose le problème de garder tous les ordinateurs synchronisés.

Dans le cadre des blockchains :

- Des clients envoient des transactions (**txs**) aux nœuds du réseau.
- Chaque nœud maintient localement une copie d'une base de données du type append only contenant une séquence de txs, c'est l'histoire du nœud.
- Synchroniser les nœuds pour qu'ils aient tous la même histoire.

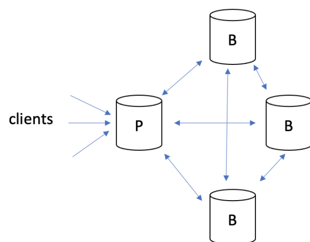
Rappel : propriétés d’une solution SMR

Un algorithme qui permet de résoudre le problème SMR doit posséder les propriétés suivantes :

- Consistency : tous les participants sont d’accord sur la même histoire (même séquences de txs).
- Liveness : Toutes les txs (valides) envoyées par les clients sont inévitablement enregistrées à un moment dans l’histoire locale des machines.

Rappel : l'architecture Primary-Backup

Une solution classique au problème SMR consiste à utiliser une architecture Primary-Backup



(1) Un nœud est désigné Primary.
(2) Le Primary propose un bloc de transactions aux autres nœuds, les Backups,
(3) Le Primary et les Backups décident (ou commit) un bloc quand ils sont tous d'accord entre eux.

Il existe de nombreuses techniques pour choisir le Primary.
Si un Primary tombe en panne, il faut en choisir un autre.

Pannes dans une architecture Primary-Backup

Dans la suite, on s'intéresse aux pannes non Byzantines (essentiellement, pertes de messages).

Dans l'architecture Primary-Backup, les pannes des backups sont faciles à gérer. Pour les pannes du primary, on distingue les cas où il tombe en panne

(1) avant l'envoi de son bloc ;
(2) après l'envoi de son bloc, mais avant de recevoir les confirmations (on dit aussi "votes") ;
(3) après la réception des votes et son commit sur le bloc.

Round

Lorsqu'un Primary tombe en panne, il faut en choisir un nouveau.

Lorsqu'on fait le choix d'un nouveau Primary, on dit qu'on “change de round.

Pour faciliter le choix d'un nouveau Primary, on numérote les rounds de manière croissante.

Étant donné un round r, on suppose pour simplifier que toutes les machines savent à l'avance quel est le Primary associé à r.

Changement de round

Seul la panne (2) pose un problème de sûreté lors d'un changement de round.

⇒ Lors du passage à un nouveau round $r + 1$, il faut s'assurer qu'aucun backup ne commit sur un bloc r si le nouveau Primary propose un bloc différent à $r + 1$

Lock-Commit

Une façon simple de garantir la sûreté est de faire en sorte que le nouveau Primary ne modifie pas la valeur du bloc envoyé lors d'un round précédent.

On utilise pour cela la technique du Lock-Commit qui consiste à :

(1) **Verrouiller un quorum avant de commiter** sur un bloc.
(2) **Vérifier un quorum avant de proposer** un nouveau bloc.

Lock-Commit

Une façon simple de garantir la sûreté est de faire en sorte que le nouveau Primary ne modifie pas la valeur du bloc envoyé lors d'un round précédent.

On utilise pour cela la technique du Lock-Commit qui consiste à :

(1) **Verrouiller un quorum avant de commiter** sur un bloc.
(2) **Vérifier un quorum avant de proposer** un nouveau bloc.

Pour cela, un nœud maintient un verrou contenant une paire (b, r), où r est le round le plus récent où un bloc b a été proposé. Un vote pour b correspond à l'envoi du verrou (b, r) par un Backup.

Lock-Commit

Une façon simple de garantir la sûreté est de faire en sorte que le nouveau Primary ne modifie pas la valeur du bloc envoyé lors d'un round précédent.

On utilise pour cela la technique du Lock-Commit qui consiste à :

(1) **Verrouiller un quorum avant de commiter** sur un bloc.
(2) **Vérifier un quorum avant de proposer** un nouveau bloc.

Pour cela, un nœud maintient un verrou contenant une paire (b, r), où r est le round le plus récent où un bloc b a été proposé. Un vote pour b correspond à l'envoi du verrou (b, r) par un Backup.

Avant de décider un bloc b dans un round r, le Primary s'assure qu'il a un quorum de $N - f$ verrous sur b dans r.

Lock-Commit

Une façon simple de garantir la sûreté est de faire en sorte que le nouveau Primary ne modifie pas la valeur du bloc envoyé lors d'un round précédent.

On utilise pour cela la technique du Lock-Commit qui consiste à :

(1) **Verrouiller un quorum avant de commiter** sur un bloc.
(2) **Vérifier un quorum avant de proposer** un nouveau bloc.

Pour cela, un nœud maintient un verrou contenant une paire (b, r), où r est le round le plus récent où un bloc b a été proposé. Un vote pour b correspond à l'envoi du verrou (b, r) par un Backup.

Avant de décider un bloc b dans un round r, le Primary s'assure qu'il a un quorum de $N - f$ verrous sur b dans r.

Lors d'un changement de Primary, le nouveau Primary recherche le plus récent bloc b ayant un quorum de $N - f$.

Primary-Backup : algorithme Lock-Commit (1/2)

État des nœuds

$Block$ = Null *# Bloc sur lequel le noeud commit*
R = 1 *# Round courant*
L = (Null,0) *# Verrou*

Code du Primary, étant donné un bloc B à propager.

send $\langle Propose, (B, R)\rangle$ to all
wait for $N - f$ distinct $\langle Lock, (b, r)\rangle$ with $b = B$ and $r = R$
send $\langle Commit, B\rangle$ to all
`terminate`

Code des Backups

wait for $\langle Propose, (b, r)\rangle$ only if $r = R$
$L := (b, r)$
send $\langle Lock, L\rangle$ to the Primary
wait for $\langle Commit, b\rangle$
$Block := b$
send $\langle Commit, Block\rangle$ to all
`terminate`

Primary-Backup : algorithme Lock-Commit (2/2)

Un changement de round peut intervenir pour plusieurs raisons :

- le temps alloué au round courant est dépassé ;
- un quorum de $N - f$ nœuds affirment que le Primary courant est en panne.

Quand un Backup passe au nouveau round r', il envoie au Primary de r' un message $\langle HighestLock, L, r' \rangle$ pour lui communiquer son verrou, puis il (re-)exécute le code Backup.

Le Primary de r' exécute le code suivant, puis le code Primary.

```
wait for N − f distinct ⟨HighestLock, (b, r1), r2⟩ if r2 = r′ = R
if b = Null in all messages then
   B := new block
else
   L := (b,r1) with the highest r1
```

Sûreté du Lock-Commit (1/2)

Théorème :

Soit r le premier round pendant lequel un nœud à commiter sur un block b. Alors, aucun Primary ne proposera un autre bloc $b' \neq b$ dans un round r' tel que $r \leq r'$.

Rappels

- On suppose un réseau asynchrone.
- On ne s'intéresse qu'aux pannes non-Byzantines, c'est-à-dire des pertes ou retards de messages (ou pannes des nœuds).

Sûreté du Lock-Commit (2/2)

La sûreté de la technique Lock-Commit est liée à la cardinalité de l'intersection des deux quorums $N - f$ dans l'algorithme.

Q_1 : le quorum $N - f$ des Backups qui ont envoyé un message $\langle Lock, (b, r) \rangle$ et commité sur b.

Q_2 : le quorum $N - f$ des Backups qui ont envoyé un message $\langle HighestLock, L, r' \rangle$, pour $r \leq r'$.

Intuition : On montre que si $|Q_1 \cap Q_2| \geq 1$, alors le bloc commité est toujours passé au prochain Primary comme la valeur commitée la plus récente, et il sera donc re-proposé par le nouveau Primary.

Lock-Commit : preuve

Soit b le bloc verrouillé et commité au round r. Par induction sur $r' \geq r$.

Lock-Commit : preuve

Soit b le bloc verrouillé et commité au round r. Par induction sur $r' \geq r$.

Cas $r' = r$. Immédiat car le Primary ne propose qu'un seul message $\langle Propose, (b, r) \rangle$ par round.

Lock-Commit : preuve

Soit b le bloc verrouillé et commité au round r. Par induction sur $r' \geq r$.

Cas $r' = r$. Immédiat car le Primary ne propose qu'un seul message $\langle Propose, (b, r) \rangle$ par round.

Cas $r' > r$. On suppose que la propriété est vraie pour tous les rounds jusqu'à r'. Considérons le changement au round $r' + 1$.

Lock-Commit : preuve

Soit b le bloc verrouillé et commité au round r. Par induction sur $r' \geq r$.

Cas $r' = r$. Immédiat car le Primary ne propose qu'un seul message $\langle Propose, (b, r) \rangle$ par round.

Cas $r' > r$. On suppose que la propriété est vraie pour tous les rounds jusqu'à r'. Considérons le changement au round $r' + 1$.

Q_1 est l'ensemble des $N - f$ Backups qui ont verrouillé b au round r, et ont donc envoyé $\langle Lock, (b, r) \rangle$ au Primary du round r.

Q_2 est l'ensemble des $N - f$ Backups qui ont envoyé $\langle HighestLock, (b', k), r' + 1 \rangle$ au Primary de $r' + 1$.

Lock-Commit : preuve

Soit b le bloc verrouillé et commité au round r. Par induction sur $r' \geq r$.

Cas $r' = r$. Immédiat car le Primary ne propose qu'un seul message $\langle Propose, (b, r) \rangle$ par round.

Cas $r' > r$. On suppose que la propriété est vraie pour tous les rounds jusqu'à r'. Considérons le changement au round $r' + 1$.

Q_1 est l'ensemble des $N - f$ Backups qui ont verrouillé b au round r, et ont donc envoyé $\langle Lock, (b, r) \rangle$ au Primary du round r.

Q_2 est l'ensemble des $N - f$ Backups qui ont envoyé $\langle HighestLock, (b', k), r' + 1 \rangle$ au Primary de $r' + 1$.

Si $Q_1 \cap Q_2 \neq \emptyset$, alors il existe un Backup p qui a envoyé $\langle Lock, (b, r) \rangle$ et $\langle HighestLock, (b', k), r' + 1 \rangle$. Par hypothèse de récurrence, pour tous les rounds $r \leq k \leq r'$, il n'y a que des messages $\langle Propose, (b, k) \rangle$. On en déduit que les verrous de tous les Primary, en particulier p, sont donc de la forme $L = (b, k)$ lors du passage à $r' + 1$, avec un round k tel que $r \leq k \leq r'$. Le bloc proposé lors du passage à $r' + 1$ est donc b.

Intersection des quorums

Étant donnés N nœuds et $f < N$ pannes, la cardinalité de l'intersection de deux sous-ensembles de $N - f$ nœuds est au moins de $N - 2f$.

- Si $N = 2f + 1$, alors l'intersection de sous-ensembles de $f + 1$ nœuds est au moins de 1 nœud.
- Si $N = 3f + 1$, alors l'intersection de sous-ensembles de $2f + 1$ nœuds est au moins de $f + 1$ nœuds.

Remarque :

Les algorithmes pour pannes non-byzantines nécessitent donc des quorums de $f + 1$ nœuds, tandis que des quorums de $2f + 1$ sont nécessaires pour garantir la sûreté des algorithmes byzantins.

Intérêts du Lock-Commit

La technique du Lock-Commit a plusieurs avantages.

- Sa sûreté est basée sur l'intersection des quorums et sa preuve ne repose pas sur une hypothèse de synchronisme.
- Aucun nœud ne peut décider avant l'étape (3) de commit.
- Il suffit de changer la taille des quorums pour passer de la tolérance de simples pannes ($f + 1$) à des pannes Byzantines ($2f + 1$).

Lock-Commit en pratique

La technique de Lock-Commit est à la base des algorithmes de consensus de nombreuses Blockchains :

- Tendermint : Cosmos
- Tenderbake : Tezos
- Hotstuff : Libra, Safestake
- Casper FFG : Ethereum
- SBFT : Ripple

Nous allons illustrer son fonctionnement à travers l'algorithme Tenderbake de la blockchain Tezos

Tenderbake

Il s'agit de l'implémentation courante de l'algorithme de consensus de Tezos.

Sa conception a été inspirée par l'algorithme Tendermint, mais avec des hypothèses plus réalistes :

- buffers bornés
- hypothèses plus faibles sur la couche de gossip (P2P)
- adapté à l'architecture Tezos.

Comme tous les algorithmes SMR, il suppose un réseau partiellement asynchrone :

- tolère jusqu'à $1/3$ de participants Byzantins ;
- l'hypothèse de synchronisme n'est utilisée que pour garantir la terminaison (aucun impact sur la sûreté).

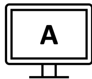
Tenderbake : principaux éléments

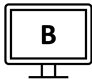
L'algorithme est exécuté avec un nombre fixe de nœuds, appelés validators ou bakers.

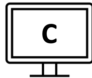
Tenderbake exécute des rounds :

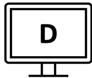
- chaque round a un Proposer choisi parmi les validators à l'aide d'un algorihtme de round-robin
- un round a 3 phases :
  - une phase **Propose** pour proposer un bloc (le proposal)
  - deux phases de vote : **Preendorse** et **Endorse**
- un quorum de preendorsements (**pQC**) nécessaire pour endorser
- un quorum d'endorsements (**eQC**) nécessaire pour décider/commiter
- l'algorithme termine si eQC est observé
- si eQC n'est pas observé, on commence le prochain round.

Tenderbake : round

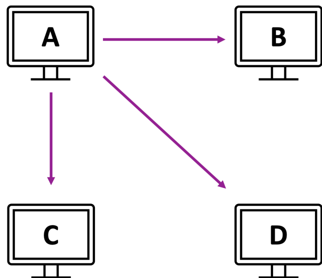






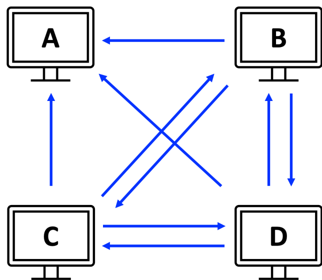


Tenderbake : round



Phase 1. Un unique baker propose un nouveau bloc (proposal) à ajouter à la blockchain. Ce bloc est propagé sur le réseau.

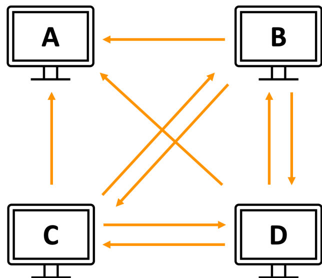
Tenderbake : round



Phase 1. Un unique baker propose un nouveau bloc (proposal) à ajouter à la blockchain. Ce bloc est propagé sur le réseau.

Phase 2. Les bakers votent (*preendorsement*) pour ce proposal et attendent de recevoir les votes des autres bakers jusqu'à atteindre un quorum (fixé à $2/3$ des votants).

Tenderbake : round

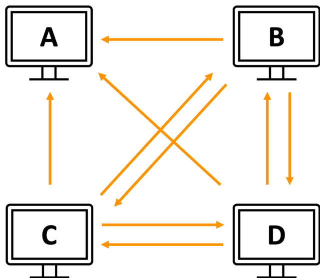


Phase 1. Un unique baker propose un nouveau bloc (proposal) à ajouter à la blockchain. Ce bloc est propagé sur le réseau.

Phase 2. Les bakers votent (*preendorsement*) pour ce proposal et attendent de recevoir les votes des autres bakers jusqu'à atteindre un quorum (fixé à $2/3$ des votants).

Phase 3. Si (et quand) un baker oberve que le quorum est atteint, il vote une seconde fois (*endorsement*) et attend le quorum (à $2/3$) sur ce 2ème vote pour valider la proposition.

Tenderbake : round



Phase 1. Un unique baker propose un nouveau bloc (proposal) à ajouter à la blockchain. Ce bloc est propagé sur le réseau.

Phase 2. Les bakers votent (*preendorsement*) pour ce proposal et attendent de recevoir les votes des autres bakers jusqu'à atteindre un quorum (fixé à $2/3$ des votants).

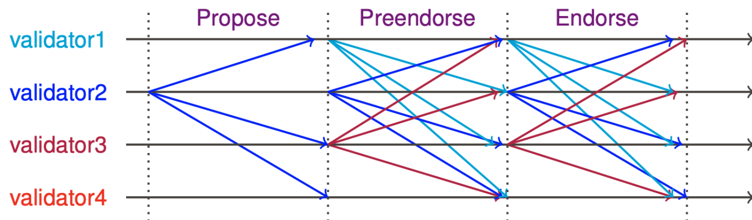
Phase 3. Si (et quand) un baker oberve que le quorum est atteint, il vote une seconde fois (*endorsement*) et attend le quorum (à $2/3$) sur ce 2ème vote pour valider la proposition.

La séquence de ces 3 phases est appelée un round.

Tenderbake : exécution dans le meilleur cas

Dans le meilleur des cas, tous les bakers corrects décident en un unique round.

- un bloc b est proposé et il est reçu par tous les bakers ;
- tous les bakers corrects preendorsent b et ils voient tous le pQC pour b : ils endorsent b ;
- ces mêmes bakers voient tous l'eQC pour b : ils décident b.



Notez que dans cette figure, le validator 4 est Byzantin : il reçoit v mais ne preendorse pas sur v.

Pourquoi deux phases de vote ? (I)

Avec un seul vote. Supposons qu'il y ait 4 bakers corrects v_1, v_2, v_3 et v_4.

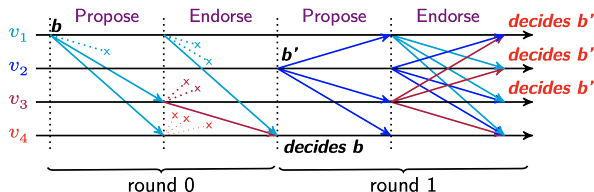
Round 0 :

- v_1 propose b, les bakers v_3 et v_4 le voient et l'endorsent
- mais seul v_4 voit l'eQC $\{v_1, v_3, v_4\}$ et décide sur b

Round 1 :

- v_2 (nouveau Proposer pour le round 1), qui n'a pas vu la proposition b précédente, propose b'
- v_1, v_2, v_3 endorsent b'
- tous les messages sont reçus et v_1, v_2, v_3 décident sur b'

La propriété d'Agreement n'est pas assurée !



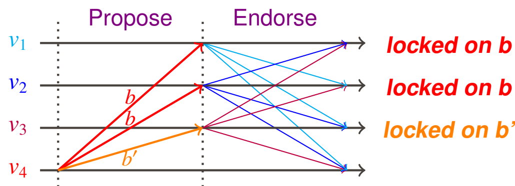
Pourquoi deux phases de vote ? (II)

Nouvelle règle : un nœud ne peut endorser qu'une fois (∼ mettre verrou sur le bloc pour lequel le nœud a voté)

Considérons alors le scénario suivant :

- v_4 est Byzantin et propose b à v_1, v_2 et b' à v_3
- v_1, v_2 endorsent b, v_3 endorse b'
- v_1, v_2, v_3 sont bloqués parce que :
  aucun d'eux ne voit un eQC, ni pour b, ni pour b'

La propriété d'Agreement n'est pas assurée !

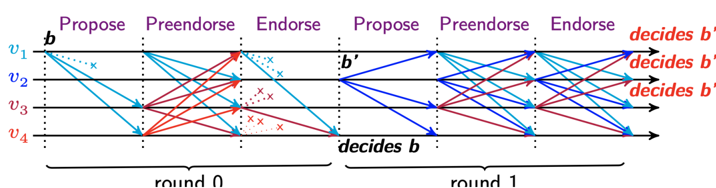


Double votes + verrous

Pour contrer cette attaque, on utilise une phase préliminaire de vote pour ne pas verrouiller un bloc qui n'a aucune chance d'être endorsé.

Phase de Preendorse :

- preendorsement : "J'envisage d'endorser b"
- un baker endorse b ssi b a un quorum de preendorsements ; ainsi il a pleinenement confiance de voter pour un bloc qui a la majorité.



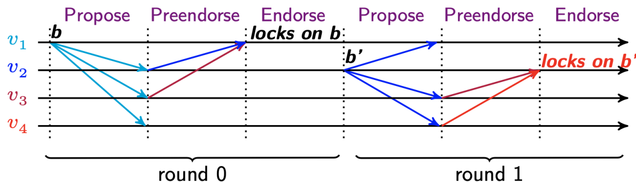
Le mécanisme de verrou est adapté pour :

- verrouiller le bloc endorsé
- seulement preendorser quand le verrou est vide

Double votes + (re)-verrouillage

Un re-verrouillage est aussi nécessaire, sinon, le scénario suivant est possible :

- v_1 verrouille b au round 0 (grâce à v_2, v_3)
- v_2 verrouille b' au round 1 (grâce à v_3, v_4)
- v_1 et v_2 doivent se réconcilier.



La propriété d'Agreement n'est toujours pas assurée !

Nouvelle règle :

- Le dernier round avec un pQC a toujours la priorité.
- Si un verrou est mis, on peut preendorser seulement sur un proposal avec un pQC d'un round plus élevé.

Tenderbake : états de bakers

i : le numéro d'un nœud
b : les transactions contenues dans un bloc
q : un quorum (= ensemble de signatures)
r : un numéro de round

Pour exécuter Tenderbake single-shot, un baker i maintient un état local qui contient :

RND_i : le round courant de i
EDR_i : un bloc endorsable
PQC_i : $\{r, q\}$ preendorsements q observé au round r
LOCK_i : $\{b, r\}$ le bloc b verrouillé au round r

On note que EDR_i, PQC_i et LOCK_i peuvent peut être vides (`null`).

Tenderbake : exécution d'un round

- Phase Propose

  ⇒ incrémentation de RND_i

  ⇒ le Proposer i envoie la proposition $\{b, \text{RND}_i, pqc\}$ avec :

  - une nouvelle valeur pour b et pqc=`null`, si EDR_i = `null`
  - une "reproposition" de $b = \text{EDR}_i$ et $pqc = \text{PQC}_i$, sinon.

- Phase Preendorse

  ⇒ à la réception d'un proposal $\{b, r, pqc\}$, un baker i preendorse b si $\text{LOCK}_i = \texttt{null}$ ou $\text{LOCK}_i.b = b$ ou $\text{LOCK}_i.r \leq pqc.r$

- Phase Endorse

  ⇒ à la réception d'un quorum q de preendorsements pour b au round courant, endorse b, $\text{EDR}_i = b$, $\text{PQC}_i = \{\text{RND}_i, q\}$ et $\text{LOCK}_i = \{\text{EDR}_i, \text{RND}_i\}$

  ⇒ à la réception d'un quorum d'endorsements pour EDR_i au round RND_i, décide sur EDR_i.

Tenderbake : exemple

Soient 4 bakers v_1, v_2, v_3, v_4 et v_3 est Byzantin

Round 1 :

- v_1 propose b
- v_1, v_2, v_3 preendorsent b
- seul v_1 voit le pQC $\{v_1, v_2, v_3\}$; il verrouille et endorse b

Tenderbake : exemple

Soient 4 bakers v_1, v_2, v_3, v_4 et v_3 est Byzantin

Round 1 :

- v_1 propose b
- v_1, v_2, v_3 preendorsent b
- seul v_1 voit le pQC $\{v_1, v_2, v_3\}$; il verrouille et endorse b

Round 2 :

- v_2 propose b'
- v_1 ne le preendorse pas car il est verrouillé sur b
- v_2, v_3, v_4 preendorsent b'
- v_4 voit le pQC $\{v_2, v_3, v_4\}$ et verrouille b'

Tenderbake : exemple

Soient 4 bakers v_1, v_2, v_3, v_4 et v_3 est Byzantin

Round 1 :

- v_1 propose b
- v_1, v_2, v_3 preendorsent b
- seul v_1 voit le pQC $\{v_1, v_2, v_3\}$; il verrouille et endorse b

Round 2 :

- v_2 propose b'
- v_1 ne le preendorse pas car il est verrouillé sur b
- v_2, v_3, v_4 preendorsent b'
- v_4 voit le pQC $\{v_2, v_3, v_4\}$ et verrouille b'

Round 3 : v_3 ne propose rien

Tenderbake : exemple

Soient 4 bakers v_1, v_2, v_3, v_4 et v_3 est Byzantin

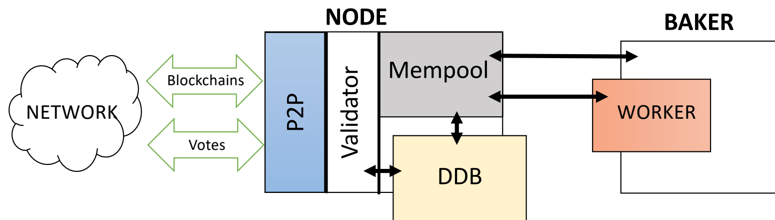
Round 1 :
- v_1 propose b
- v_1, v_2, v_3 preendorsent b
- seul v_1 voit le pQC $\{v_1, v_2, v_3\}$; il verrouille et endorse b

Round 2 :
- v_2 propose b'
- v_1 ne le preendorse pas car il est verrouillé sur b
- v_2, v_3, v_4 preendorsent b'
- v_4 voit le pQC $\{v_2, v_3, v_4\}$ et verrouille b'

Round 3 : v_3 ne propose rien

Round 4 :
- v_4 propose b' (verrouillé) avec un round de pQC égal à 2
- v_1 preendorse b' car son round de pQC (=2) est plus grand que celui du round qu'il a verrouillé (=1)
- v_2, v_4 preendorsent b' également
- v_1, v_2, v_4 voient le pQC $\{v_1, v_2, v_4\}$ et endorsent b'
- v_1, v_2, v_4 voit l'eQC $\{v_1, v_2, v_4\}$ et décident sur b'

Architecture simplifiée de Tezos



Fonctionnement d'un nœud

- communique et échange des blockchains (complètes ou seulement les têtes) avec les autres nœuds
- Maintient la meilleure version de la blockchain qu'il a obtenu
- Passe les opérations reçues aux bakers avev qui il est connecté

Fonctionnement d'un baker

- reçoit les deux blocs de tête de la blockchain du Mempool (seuls 2 bocs sont nécessaires dans Tenderbake)
- implémente l'algorithme de consensus pour décider de voter (ou non) sur cette tête
- récupère les votes envoyés par les autres nœuds dans le Mempool pour vérifier l'obtention d'un quorum

Consensus et comités

Les solutions pour résoudre le problème SMR dans les blockchains se différencient selon deux critères :

- Le mécanisme pour sélectionner les utilisateurs qui sont autorisés à proposer/voter des blocs
- L'algorithme de consensus pour converger vers une décision.

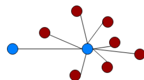
Cependant, ces mécanismes et algorithmes sont très sensibles aux attaques qui consistent à maîtriser la topologie du réseau P2P.

Attaques (I)

Les différentes technique pour choisir un Primary doivent être suffisamment sécurisées pour ne pas être impactées par différentes attaques bien connues.

Une de ces attaques est l'attaque Sybil où l'adversaire essaie de créer autant d'identités qu'il peut pour gagner de l'influence sur le réseau.

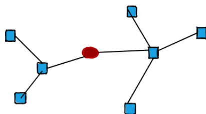
Par exemple, il peut essayer d'encercler un nœud comme ceci :



Dans cette configuration, il est difficile de s'assurer que l'algorithme de sélection de Primary ne sélectionne pas toujours le même acteur (qui possède en réalité plusieurs nœuds).

Attaques (II)

Une autre attaque est l'attaque Eclipse où un attaquant cherche à couper le réseau en deux parties en forçant un point de passage obligé qu'il est le seul à maîtriser, comme ci-dessous :



Dans cette configuration, il est impossible de déployer une blockchain cohérente car les deux parties vont nécessairement diverger en créant deux mondes distincts.

Pour cette raison, on suppose souvent qu'entre deux nœuds honnêtes, il existe toujours un chemin passant uniquement par des nœuds honnêtes.

Comment éviter les attaques éclipses ?

Protocole de découverte des peers :

- au moment de démarrer, un nœud se connecte à une liste bien connue (par ex. maintenue par une fondation) de peers auxquels il peut faire confiance ;
- une fois connecté, le nœuds demande à ses peers de lui communiquer une (sous-)liste de leurs peers ;
- en cas de problème (censure, etc.), on peut lancer un nœud en lui précisant manuellement une (ou plusieurs) adresse(s) de peets auxquels il doit se connecter.

Comment choisir un Primary ?

La sélection d'un nœud “Primary” pour proposer et diffuser un bloc peut se faire de plusieurs manières :

- Proof of work
- Proof of stake
- Proof of history, space, authority...

Proof of work (PoW)

En faisant en sorte que les candidats résolvent un problème difficile (mais pas infaisable) sur le plan informatique, cette technique permet de s'assurer que la possibilité de devenir Primary est rare.

Le minage du Bitcoin consiste à trouver une solution à l'équation $hash(B) \leq T$ pour laquelle aucune technique, sauf la force brute, est connue.

Ce schéma date du début des années 1990. Voir par exemple l'article de Dwork & Naor datant de 1993.

La valeur de la constante T est mise à jour de sorte qu'il n'y ait qu'un seul bloc qui soit produit dans un certain intervalle de temps. Dans Bitcoin, cet intervalle est de 10 minutes. Cette valeur semble avoir été fixée pour limiter la croissance de la taille de la blockchain, en partculier pour les clients légers (S. Nakamoto).

PoW : avantages et inconvénients

Avantages

- empêche les attaques Sybil car la création de nouvelles identités nécessite plus de puissance de calcul ;
- limite fortement les phénomènes de double dépense car les blocs sont générées très lentement, ce qui limite les forks de la blockchain.

Inconvénients

- Consommation énergétique
- Centralisation à cause de la création de “fermes de minage” et l’accumulation des richesses par certains acteurs.

PoS (Proof of Stake)

La Proof of Stake, en Français preuve d'enjeu, est une technique pour constituer les comités de votes/propositions.

Seuls les utilisateurs qui ont *accumulé* suffisamment de tokens peuvent proposer des blocs et voter (la somme varie selon les blockchains).

one-cpu-one-vote $\rightsquigarrow$ one-coin-one-vote

Un utilisateur qui propose/vote met en *dépôt* une partie de son argent. Cette somme lui est retirée s'il se comporte mal.

L'algorithme établie un ordre sur les mineurs afin de déterminer qui commence à proposer un bloc, puis le suivant, etc.

PoS : avantages et inconvénients

Avantages

- Résistant aux attaques Sybil car un Byzantin doit avoir beaucoup d'argent pour attaquer la chaîne. Par ailleurs, plus on possède d'argent, moins on a d'intérêts à tricher.
- Pas de dépenses d'énergie inutile, peu inflationniste.

Inconvénients

- L'argent mis de côté pour miner n'est pas utilisé (thésaurisation).
- Seuls les "riches" peuvent miner.
- *Nothing-at-stake*

DPoS : Delegated Proof of Stake

Variante de la preuve d’enjeu où les détenteurs de tokens peuvent élire des délégués pour valider des blocs en leur nom.

Cela permet d’éliminer le problèmes “seuls les riches peuvent miner”.

Pour être élu, un délégué doit bien faire son travail et reverser une part de ce qu’il a gagné à ses électeurs.

Avantages

- Pas de dépense d’énergie inutile, peu inflationniste, consensus encore plus rapide que POS car moins de valideurs.

Inconvénients

- Risque de centralisation (si peu de délégués), collusion

LPoS : Liquid Proof of Stake

Variante de la DPoS qui permet un acteur de déléguer ses droits de baking (proportionnels aux tokens possédés) sans que les délégués aient la garde de ses tokens.

Le principal intérêt de cette variante est que les acteurs qui délèguent peuvent récupérer leurs tokens en dépôt à tout moment (contrairement au mécanisme dans Ethereum).

PoA : Proof of Authority

La liste des miners est connue à l'avance, de manière “autoritaire”. Ces miners sont connus et affichés publiquement sur la blockchain et leur réputation (et non la puissance ou la richesse) est mise en jeu s'ils se comportent mal.

Avantages

- Efficacité énergétique, consensus très rapide.

Inconvénients

- Autorité centrale, perte de décentralisation.

Streamlet : consensus pour synchronisme partiel

Algorithme de consensus pour blockchain inventé par B. Chan et E. Shi en 2020.

- Protocole avec un comité (*permissioned*) de N participants.
- $f < N/3$ Byzantins.
- Horloges synchronisées (epochs).
- Propriétés garanties : consistency et liveness

Streamlet : le protocole

Protocole à 2 phases (Propose et Vote) + critére de finalisation.

On dit qu'un bloc est notarisé s'il a reçu plus de $2N/3$ votes. Par extension, une chaîne est notarisée quand elle ne contient que des blocs notarisés.

Un bloc est finalisé quand on est certain qu'il ne pourra plus jamais être sorti de la chaîne.

Notarisé $\neq$ finalisé.

Streamlet : phase 1 – Propose

Phase – Propose

Au début de l'*epoch* e, le leader de e propose un nouveau bloc b qui étend la plus longue chaîne notarisée qu'il ait vue.

Le bloc b qui est envoyé est signé par le leader de l'*epoch*.

Streamlet : phase 2 – Vote

Phase – Vote

Pour chaque *epoch* e, chaque nœud n du comité vote :

- une seule fois dans l'*epoch* e
- uniquement pour le premier bloc b proposé par le leader de e
- et si b étend la plus longue chaîne notarisée de n.

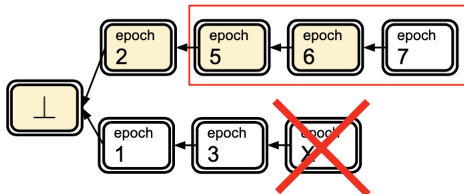
Un vote est simplement un message contenant la signature de n pour le bloc b.

Streamlet : finalisation

Aux 2 phases précédentes s'ajoute un mécanisme de finalisation.

Lorsque 3 blocs adjacents b_1, b_2 et b_3 d'une chaîne notarisée ont des numéros d'*epoch* consécutifs, alors la chaîne peut être considérée comme finale jusqu'à b_2 (inclus).

Streamlet : exemple



Streamlet : consistency (1/3)

Idée de la preuve.

Sur l'exemple précédent, on montre qu'il ne peut y avoir un autre bloc notarisé avec une distance à *genesis* égale à celle du bloc epoch-6.

Lemme : Pour chaque epoch e, il y a au plus un seul bloc notarisé.

Streamlet : consistency (1/3)

Idée de la preuve.

Sur l'exemple précédent, on montre qu'il ne peut y avoir un autre bloc notarisé avec une distance à *genesis* égale à celle du bloc epoch-6.

Lemme : Pour chaque epoch e, il y a au plus un seul bloc notarisé.

Supposons qu'il existe 2 blocs b and b' notarisés dans l'epoch e. Il existe donc 2 ensembles S et S' d'au moins $2N/3$ participants qui ont voté respectivement pour b et b'. Puisqu'il y a N participants, $|S \cap S'| \geq N/3$, et donc il existe au moins un participant honnête p dans $S \cap S'$. D'après le protocole, p ne peut voter que pour 1 bloc, donc $b = b'$ $\square$.

Streamlet : consistency (1/3)

Idée de la preuve.

Sur l'exemple précédent, on montre qu'il ne peut y avoir un autre bloc notarisé avec une distance à *genesis* égale à celle du bloc epoch-6.

Lemme : Pour chaque epoch e, il y a au plus un seul bloc notarisé.

Supposons qu'il existe 2 blocs b and b' notarisés dans l'epoch e. Il existe donc 2 ensembles S et S' d'au moins $2N/3$ participants qui ont voté respectivement pour b et b'. Puisqu'il y a N participants, $|S \cap S'| \geq N/3$, et donc il existe au moins un participant honnête p dans $S \cap S'$. D'après le protocole, p ne peut voter que pour 1 bloc, donc $b = b'$ $\square$.

D'après ce lemme, un bloc en conflit avec celui de l'epoch 6 est donc d'epoch (strictement) inférieure ou supérieure à 6.

Streamlet : consistency (2/3)

Supposons que l'epoch de X soit < 5. Puisque qu'il y a au moins $2N/3$ votes pour X, au moins $N/3$ votes sont honnêtes. De plus, ces participants honnêtes ont notarisé le bloc epoch-3 et ils ne peuvent plus voter pour le bloc epoch-5 (car il n'étend pas la plus longue chaîne). Par conséquent, comme il n'y a que $f < N/3$ Byzantins, il ne peut y avoir $\geq 2N/3$ votes pour le bloc epoch-5. Contradiction $\square$.

Streamlet : consistency (2/3)

Supposons que l'epoch de X soit < 5. Puisque qu'il y a au moins $2N/3$ votes pour X, au moins $N/3$ votes sont honnêtes. De plus, ces participants honnêtes ont notarisé le bloc epoch-3 et ils ne peuvent plus voter pour le bloc epoch-5 (car il n'étend pas la plus longue chaîne). Par conséquent, comme il n'y a que $f < N/3$ Byzantins, il ne peut y avoir $\geq 2N/3$ votes pour le bloc epoch-5. Contradiction □.

Supposons que l'epoch de X soit > 7. Puisque le bloc epoch-7 est notarisé, il y a au moins $N/3$ participants honnêtes qui ont notarisé le bloc epoch-6. Ces participants ne peuvent plus voter pour X (plus longue chaîne). Par conséquent, comme il n'y a que $f < N/3$ Byzantins, il ne peut y avoir $\geq 2N/3$ votes pour le bloc X. Contradiction □.

Streamlet : consistency (3/3)

On peut facilement généraliser l'idée précédente.

Lemme. Si un participant honnête a une chaîne notarisée contenant 3 blocs adjacents b_0, b_1, b_2 avec des numéros d'epoch consécutifs e, $e + 1$, and $e + 2$, alors il ne peut y avoir de bloc $b \neq b_1$ qui soit notarisé avec la même longueur que b_1.

Streamlet : consistency (3/3)

On peut facilement généraliser l'idée précédente.

Lemme. Si un participant honnête a une chaîne notarisée contenant 3 blocs adjacents b_0, b_1, b_2 avec des numéros d'epoch consécutifs e, $e+1$, and $e+2$, alors il ne peut y avoir de bloc $b \neq b_1$ qui soit notarisé avec la même longueur que b_1.

Théorème (Consistency). Soit ch et ch', deux chaînes notarisées se terminant avec 3 blocs avec des numéros d'epoch consécutifs. Si l et l' sont les longueurs de ch et ch', respectivement. De plus, si $l \leq l'$ alors $ch[: l-1]$ est un préfixe de $ch'[: l'-1]$.

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.

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, col₀ ∈ {R, B, ⊥})
5:      col := col₀ // 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)
```

$\mathcal{C}$ 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$_0 \in \{\mathtt{R}, \mathtt{B}, \bot\}$)
2: col := col$_0$, cnt := 0
3: **while** undecided **do**
4: **if** col $= \bot$ **then continue**
5: $\mathcal{K}$:= SAMPLE($\mathcal{N} \backslash u, k$)
6: P := [QUERY(v, col) **for** $v \in \mathcal{K}$]
7: **for** col$' \in \{\mathtt{R}, \mathtt{B}\}$ **do**
8: **if** P.COUNT(col$'$) $\geq \alpha \cdot k$ **then**
9: **if** col$' \neq$ col **then**
10: col := col$'$, cnt := 0
11: **else**
12: **if** ++cnt $> \beta$ **then** ACCEPT(col)

Snowball

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

1: **procedure** SNOWBALLLOOP(u, col$_0 \in \{\mathtt{R}, \mathtt{B}, \perp\}$)
2: col := col$_0$, lastcol := col$_0$, cnt := 0
3: $d[\mathtt{R}] := 0$, $d[\mathtt{B}] := 0$
4: **while** undecided **do**
5: **if** col $= \perp$ **then continue**
6: $\mathcal{K} :=$ SAMPLE($\mathcal{N} \backslash u, k$)
7: $P :=$ [QUERY(v, col) **for** $v \in \mathcal{K}$]
8: **for** col$'$ $\in \{\mathtt{R}, \mathtt{B}\}$ **do**
9: **if** P.COUNT(col$'$) $\geq \alpha \cdot k$ **then**
10: d[col$'$]++
11: **if** d[col$'$] $>$ d[col] **then**
12: col := col$'$
13: **if** col$'$ $\neq$ lastcol **then**
14: lastcol := col$'$, cnt := 0
15: **else**
16: **if** ++cnt $> \beta$ **then** ACCEPT(col)