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

Briques technologiques

L'implémentation d'une blockchain repose sur trois briques techologiques :

- cryptographie
- registre distribué
- protocole de consensus

Protocole de consensus

Pour s'accorder sur la liste de blocs à inclure dans le registre, les ordinateurs formant la blockchain utilisent un protocole de consensus (distribué)

Exemples :

- Bitcoin : Proof-of-Work et la règle de la plus longue chaîne
- Ethereum : Proof-of-Stake et algorithme BFT (Byzantine Fault Tolerant)

BITCOIN

Origines

Décrit en 2008 par un certain Satoshi Nakamoto dans l'article

Bitcoin : A peer-to-peer electronic cash system

Bitcoin vise à construire un système distribué de paiement, sans tiers de confiance, pour un nombre quelconque de participants

La principale difficulté est de concevoir un protocole de consensus permettant à tous les participants de se mettre d'accord, sans connaître le nombre de participants

Le problème du consensus distribué avec un nombre connu de participants a été étudié depuis la fin des années 70, notamment à travers le problème des généraux Byzantins (voir plus loin)

Doubles dépenses

Contrairement à une monnaie physique, une pièce de monnaie électronique peut être copiée sans effort, ce qui permet potentiellement de dépenser plusieurs fois la même pièce.

Ce problème ne se pose pas quand la monnaie est gérée d'une manière centralisée par une banque (qui est la seule à tenir les comptes et autoriser ou non des transactions).

Dans un système distribué (ou décentralisé), une solution pour résoudre ce problème est que les participants vote pour se mettre d'accord sur l'état des comptes après chaque transaction.

Attaque Sybil

Le problème d'un système de vote avec un nombre quelconque de participants est que certains peuvent tenter de tricher en créant plusieurs identités, afin d'accroître leur pouvoir de vote

Ce type d'attaque est connu sous le nom d'attaque sybil.

Proof-of-Work

La solution du Bitcoin pour résoudre la double dépense et l'attaque sybil est le Proof-of-Work (PoW)

PoW : Un bloc est valide si son empreinte (hash) commence par un certain nombre de zéros. Cette condition ne peut être satisfaite que par force brute, ce qui rend la production d'un bloc très consommatrice en calculs (donc en énergie)

En Proof-of-Work, un vote = un CPU

Ainsi, multiplier les identités dans ce système revient à disposer de plus de pouvoir de calcul, ce qui entraîne un coût élevé

Plus longue chaîne

Que se passe-t-il si deux participants produise un bloc en même temps ?

Ce problème est lié à celui de la double dépense, puisque pour dépenser deux fois le même bitcoin, il suffit de produire deux blocs avec le même parent.

On parle de fork dans la blockchain si deux blocs ont le même parent

Lorsqu'il existe plusieurs branches dans Bitcoin, la règle pour ajouter un nouveau bloc consiste à choisir la chaîne formée par la la branche la plus longue

Le consensus de Nakamoto

Cette combinaison astucieuse de PoW et de la règle de la chaîne la plus longue est connue sous le nom de consensus de Nakamoto

La chaîne avec la plus grande preuve de travail accumulée (c'est-à-dire la plus longue chaîne) est considérée comme valide

À mesure que les blocs sont ajoutés et deviennent plus anciens, ils deviennent de moins en moins susceptibles d'être annulés

Ce concept est connu sous le nom de finalité probabiliste, où un bloc finalisé est un bloc qui restera définitivement dans la chaîne

ETHEREUM

Origines

Créée en 2014 par Vitalik Buterin

Ethereum : A next-generation smart contract and decentral- ized application platform

Ethereum étend les fonctionnalités de Bitcoin avec l'introduction des smart-contracts

Bien qu'Ethereum ait été implémentée à l'origine avec un consensus de Nakamoto, l'idée départ était de passer à une autre technologie de consensus appelée Proof-of-Stake

Proof-of-Stake

Le concept de Proof-of-Stake (PoS) date de 2011. Il a été posté sur le forum BitcoinTalk

"I am wondering if as bitcoins become more widely distributed, whether a transition from a proof of work based system to a proof of stake one might happen. What I mean by proof of stake is that instead of your "vote" on the accepted transaction history being weighted by the share of computing resources you bring to the network, it's weighted by the number of bitcoins you can prove you own, using your private keys."

Dans PoS, le pouvoir de vote n'est pas déterminé par les ressources informatiques (comme dans PoW), mais par le nombre de pièces numériques que possède un participant

En Proof-of-Stake, 1 pièce = 1 vote

BFT

PoS soulève de nombreuses questions :

- Est-ce que tous les participants peuvent proposer des blocs ? Sinon, comment choisir un sous-ensemble ?
- Si c'est uniquement celui qui a le plus d'argent qui propose, alors ce n'est plus un système ouvert ? Comment changer ?
- Comment faire en sorte qu'il n'y ait qu'un seul producteur à un instant donné ?

PoS permet de passer d'un monde ouvert à un monde fermé (en restreignant les participants qui peuvent proposer des blocs)

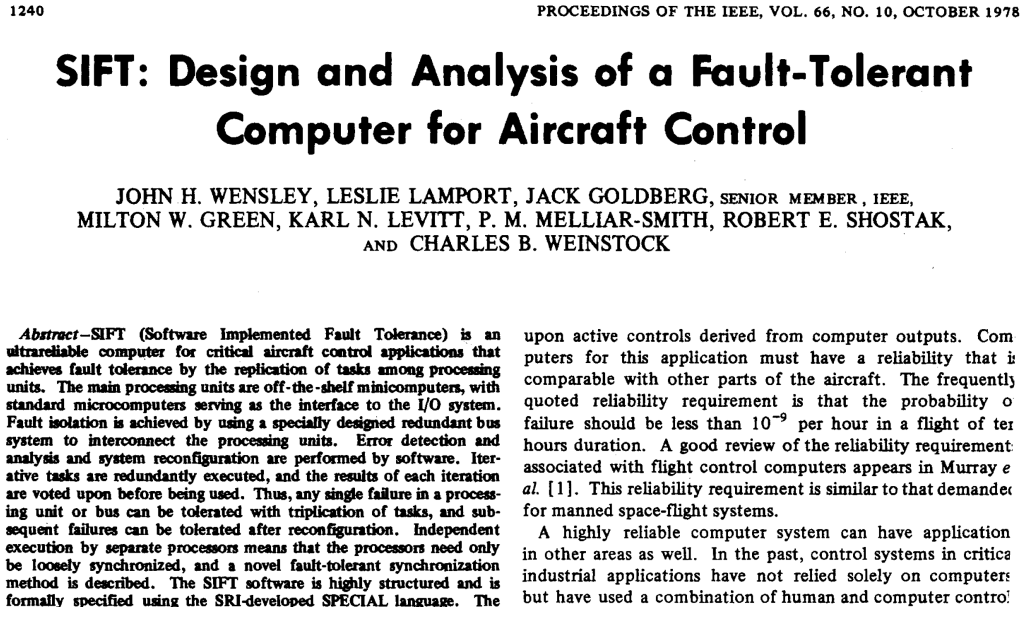
Passer à un monde fermé permet de ré-utiliser des idées sur les protocoles de consensus distribués avec un nombre connu de participants (travaux débutés dans les années 80). Il s'agit des protocoles de la famille **BFT** (Byzantin Fault Tolerant)

LE PROBLÈME DU CONSENSUS DISTRIBUÉ

Historique (1/2)

Le problème du consensus est apparu bien avant le Bitcoin.

- Dans les années 70, on a commencé à utiliser des ordinateurs pour les commandes de vol des avions
- Au début des années 80, la NASA a lancé le projet SIFT (Software Implemented Fault Tolerance) pour un ordinateur d'avion tolérant aux pannes.



1240 PROCEEDINGS OF THE IEEE, VOL. 66, NO. 10, OCTOBER 1978

SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control

JOHN H. WENSLEY, LESLIE LAMPORT, JACK GOLDBERG, SENIOR MEMBER, IEEE, MILTON W. GREEN, KARL N. LEVITT, P. M. MELLIAR-SMITH, ROBERT E. SHOSTAK, AND CHARLES B. WEINSTOCK

Abstract—SIFT (Software Implemented Fault Tolerance) is an ultrareliable computer for critical aircraft control applications that achieves fault tolerance by the replication of tasks among processing units. The main processing units are off-the-shelf minicomputers, with standard microcomputers serving as the interface to the I/O system. Fault isolation is achieved by using a specially designed redundant bus system to interconnect the processing units. Error detection and analysis and system reconfiguration are performed by software. Iterative tasks are redundantly executed, and the results of each iteration are voted upon before being used. Thus, any single failure in a processing unit or bus can be tolerated with triplication of tasks, and subsequent failures can be tolerated after reconfiguration. Independent execution by separate processors means that the processors need only be loosely synchronized, and a novel fault-tolerant synchronization method is described. The SIFT software is highly structured and is formally specified using the SRI-developed SPECIAL language. The

upon active controls derived from computer outputs. Com puters for this application must have a reliability that i comparable with other parts of the aircraft. The frequently quoted reliability requirement is that the probability o failure should be less than 10^{-9} per hour in a flight of ter hours duration. A good review of the reliability requirement associated with flight control computers appears in Murray *e al.* [1]. This reliability requirement is similar to that demande for manned space-flight systems.

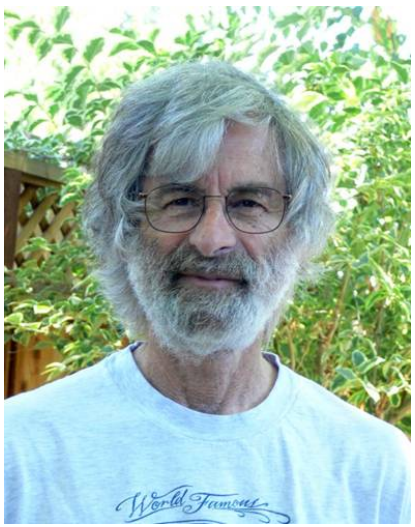
A highly reliable computer system can have application in other areas as well. In the past, control systems in critica industrial applications have not relied solely on computers but have used a combination of human and computer contro

- Le problème du consensus Byzantin a été conçu et formalisé par R. Shostak, chercheur au SRI International.

Historique (2/2)

Depuis ces années, le problème a été très largement étudié (lien).

Parmi les algorithmes les plus utilisés, on trouve Raft et Paxos



Une des personnes ayant largement contribuée à ce domaine est Leslie Lamport, Turing Award 2013.

Le problème SMR

Dans la littérature, 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.

Le problème SMR

Dans la littérature, 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.

Protocole distribué

Une solution au problème SMR prend la forme d'un protocole.

Il s'agit d'un algorithme exécuté par chaque machine pour :

- maintenir un état local ;
- effectuer des calculs qui dépendent de l'état local de la machine, et peuvent le modifier ;
- recevoir des messages des autres machines ou clients ;
- envoyer des messages aux autres machines.

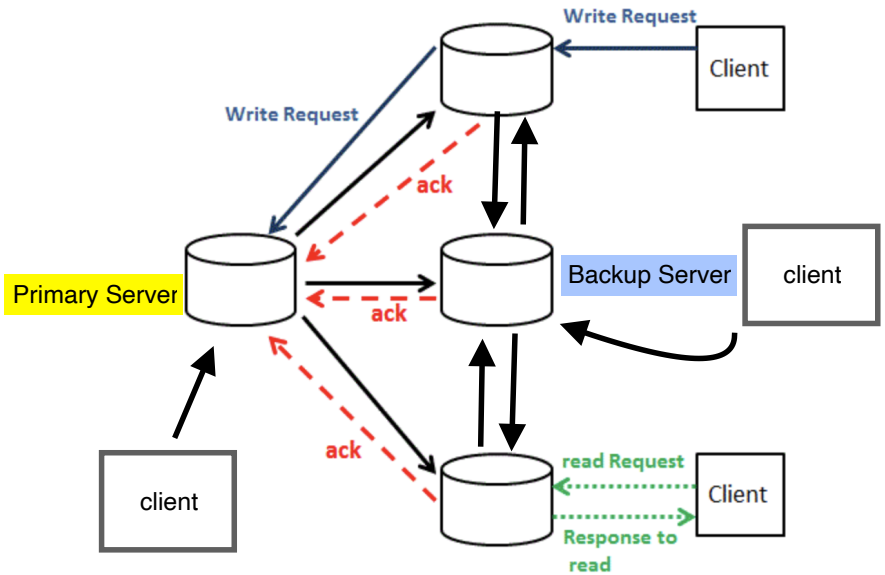
Propriétés d'un algorithme SMR

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.

SMR : Primary-Backup (1/2)

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



Un nœud est désigné comme le leader et les autres les backups.

Les commandes d'écriture sont envoyées au leader qui se charge de les propager (broadcast) aux autres nœuds (backups) et de réaliser un consensus.

Les lectures sont traitées localement.

Le leader regroupe également les commandes d'écriture dans des listes qu'il envoie aux backups (ces listes sont comme les blocs d'une blockchain).

SMR : Primary-Backup (2/2)

Sous les hypothèses suivantes, l'architecture Primary-Backup permet de résoudre le problème SMR.

(H1) Honnêteté : tous les nœuds suivent le protocole, sans en dévier.

(H2) Réseau fermé : tous les nœuds qui participent au consensus sont connus de tous et à l'avance.

(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.

On suppose aussi une infrastructure à clés publiques (PKI) pour garantir la confidentialité des communication, l'authentification des participants, l'intégrité et la non-répudiation des messages.

Pour choisir le leader, les nœuds peuvent utiliser une stratégie du type **round-robin**. Si un leader tombe en panne, il est facile d'en choisir un autre.

Pannes dans une architecture Primary-Backup

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.

$\Rightarrow$ 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

Questions

Si ce problème est très ancien, connu, et que des solutions existent, pourquoi est-il encore nécessaire de faire de la recherche dans ce domaine ?

En quoi les algorithmes implémentés dans les blockchains sont-ils différents des solutions connues ?

Consensus distribué dans les blockchains

Les solutions au problème du consensus proposées dans les blockchains doivent prendre en compte les spécificités suivantes :

Consensus distribué dans les blockchains

Les solutions au problème du consensus proposées dans les blockchains doivent prendre en compte les spécificités suivantes :

- Fautes Byzantines : les nœuds de la blockchain peuvent provoquer des fautes arbitraires : être en panne, envoyer des messages arbitraires, omettre d'en envoyer, etc. Bref, ne pas suivre les règles du jeu !

Consensus distribué dans les blockchains

Les solutions au problème du consensus proposées dans les blockchains doivent prendre en compte les spécificités suivantes :

- Fautes Byzantines : les nœuds de la blockchain peuvent provoquer des fautes arbitraires : être en panne, envoyer des messages arbitraires, omettre d'en envoyer, etc. Bref, ne pas suivre les règles du jeu !
- Système ouverts : n'importe qui peut décider de participer, on ne connait pas nécessairement tous les participants.

Consensus distribué dans les blockchains

Les solutions au problème du consensus proposées dans les blockchains doivent prendre en compte les spécificités suivantes :

- Fautes Byzantines : les nœuds de la blockchain peuvent provoquer des fautes arbitraires : être en panne, envoyer des messages arbitraires, omettre d'en envoyer, etc. Bref, ne pas suivre les règles du jeu !
- Système ouverts : n'importe qui peut décider de participer, on ne connait pas nécessairement tous les participants.
- Aucune informations globales disponibles

Consensus distribué dans les blockchains

Les solutions au problème du consensus proposées dans les blockchains doivent prendre en compte les spécificités suivantes :

- Fautes Byzantines : les nœuds de la blockchain peuvent provoquer des fautes arbitraires : être en panne, envoyer des messages arbitraires, omettre d'en envoyer, etc. Bref, ne pas suivre les règles du jeu !
- Système ouverts : n'importe qui peut décider de participer, on ne connait pas nécessairement tous les participants.
- Aucune informations globales disponibles
- Réseau P2P : les blockchains reposent sur un réseau non structuré, avec perte de messages potentielles, temps de propagation des données non connus (et non borné), etc.

Relâchement des hypothèses

Pour résoudre le problème SMR dans les blockchains, il faut réussir à relâcher les hypothèses H1, H2 et H3.

Dans la suite de ce cours, on commence par relâcher l'hypothèse H1 (Honnêteté).

(H'1) : on suppose qu'il existe au plus f nœuds Byzantins parmi les N participants, avec $f < N$.

$\Rightarrow$ Sous cette hypothèse (H'1), le consensus doit être "réussi" pour les nœuds honnêtes

Modèle synchrone

L'hypothèse **(H3)** sur les communications synchrones peut être reformulée de la manière suivante :

Le temps est découpé en rounds, d'une durée égale à Δ et les nœuds progressent de manière synchrone (lock step). Pendant ce temps, tous les messages envoyés au début d'un round ont le temps d'arriver à destination pour le début du round suivant.

Exercice 1

Sync HotStuff: Simple and Practical Synchronous State Machine Replication

Ittai Abraham[1], Dahlia Malkhi[2]*, Kartik Nayak[3]*, Ling Ren[4]*and Maofan Yin[5]*

[1]VMware Research iabraham@vmware.com
[2]Calibra dahliamalkhi@gmail.com
[3]Duke University kartik@cs.duke.edu
[4]University of Illinois at Urbana-Champaign renling@illinois.edu
[5]Cornell University tedyin@cs.cornell.edu

***Abstract*—Synchronous solutions for Byzantine Fault Tolerance (BFT) can tolerate up to minority faults. In this work, we present Sync HotStuff, a surprisingly simple and intuitive synchronous BFT solution that achieves consensus with a latency of 2Δ in the steady state (where Δ is a synchronous message delay upper bound). In addition, Sync HotStuff ensures safety in a weaker synchronous model in which the synchrony assumption does not have to hold for all replicas all the time. Moreover, Sync HotStuff has optimistic responsiveness, i.e., it advances at network speed when less than one-quarter of the replicas are not responding. Borrowing from practical partially synchronous BFT solutions, Sync HotStuff has a two-phase leader-based structure, and has been fully prototyped under the standard synchrony assumption. When tolerating a single fault, Sync HotStuff achieves a throughput of over 280 Kops/sec under typical network performance, which is comparable to the best known partially synchronous solution.**

I. Introduction

Byzantine Fault Tolerance (BFT) protocols relying on a synchrony assumption have the advantage of tolerating up to one-half Byzantine faults [1], while asynchronous or partially synchronous protocols tolerate only one-third [2]. On the flip side, synchronous protocols are often considered impractical for three main reasons. First, existing synchronous protocols require a large number of rounds. Second, most synchronous protocols require lock-step execution (i.e., replicas must start

Sync HotStuff tolerates up to one-half Byzantine replicas. Second, inspired by Hanke et al. [3], Sync HotStuff does not require lock-step execution in the steady state. Third, with minor modifications, Sync HotStuff can handle a weaker and more realistic synchrony model suggested by Chan et al. [4]. Finally, Sync HotStuff is prototyped and shown to offer practical performance. It achieves a throughput comparable to partially synchronous protocols and the commit latency is roughly a single maximum round-trip delay. Given the above properties, we believe Sync HotStuff can be the protocol of choice for single-datacenter replicated services as well as consortium blockchain applications.

We proceed to elaborate on the key techniques and key results of Sync HotStuff, which removes performance barriers on synchronous BFT under weaker assumptions.

Near-optimal latency. The first key contribution is the aforementioned extremely simple steady state protocol (Figure 1). We observe that waiting for a single maximum round-trip delay suffices for replicas to commit. Furthermore, our protocol does not have to be executed in a lock-step fashion, despite relying on synchrony. In other words, other than the concurrent waiting step, replicas move to the next step upon receiving enough messages of the previous step, without waiting for