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

Sélection des mineurs et protocole de consensus

On entend souvent parler de Proof-of-X et d'algorithme de consensus.

Est-ce que vous savez faire la différence entre les deux concepts ?

Sélection des mineurs et protocole de consensus

On entend souvent parler de Proof-of-X et d'algorithme de consensus.

Est-ce que vous savez faire la différence entre les deux concepts ?

Ce sont en fait deux mécanismes différents, qui n'accomplissent pas la même fonction dans une blockchain.

Ils travaillent ensemble, mais ils ne sont ni équivalents ni interchangeables.

Proof-of-X

Proof-of-X est mécanisme de sélection du prochain proposeur de bloc.

Il répond à la question : Qui a le droit de proposer le prochain bloc ?

Dans une blockchain ouverte (permissionless), on doit empêcher qu'un seul acteur produise tous les blocs.

Les Proof-of-X sont donc des mécanismes économiques/probabilistes, pas des algorithmes de consensus :

- Proof-of-Work $\rightarrow$ celui qui dépense le plus d'énergie a le plus de chance d'être choisi.
- Proof-of-Stake $\rightarrow$ celui qui verrouille le plus de jetons a le plus de chance d'être choisi.

Ils servent principalement à décourager les attaques (coût élevé), rendre la sélection pseudo-aléatoire, empêcher le spam.

Ils ne garantissent pas à eux seuls qu'un seul état sera accepté par tous.

Algorithmes de consensus

C'est la manière dont les participants s'accordent sur un unique historique (quelle branche de la blockchain est la bonne)

Il répond à la question : Quel est le bon bloc accepté par tout le réseau ?

Dans les blockchains, cet accord s'obtient de deux façons selon la technologie :

Algorithmes de consensus

C'est la manière dont les participants s'accordent sur un unique historique (quelle branche de la blockchain est la bonne)

Il répond à la question : Quel est le bon bloc accepté par tout le réseau ?

Dans les blockchains, cet accord s'obtient de deux façons selon la technologie :

- **Nakamoto** : c'est la règle de la plus longue chaîne ou ses variantes (GHOST, NG). C'est un consensus probabiliste où on accepte un bloc avec une confiance croissante selon la profondeur.
- **Consensus BFT** (PBFT, Tendermint, Hotstuff, Casper/FFG) : ce sont des algorithmes déterministes qui permettent de finaliser un bloc si 2/3 des validateurs le signent. Ces consensus assurent que tous les participants sont d'accord sur quel bloc est valide, quel fork est choisi, quel état final est partagé.

PoX vs consensus

Pourquoi il faut absolument distinguer PoX et algorithmes de consensus ?

(1) Ils adressent deux problèmes différents :

- PoX = comment choisir un proposeur honnête sans permission ?
- Consensus = comment éviter les forks définitivement ?

(2) Une blockchain peut changer l'un sans changer l'autre

Par ex. Ethereum a gardé son consensus BFT même en changeant son PoW par PoS. De même, Bitcoin pourrait changer son consensus (ex : GHOST, NG,...) sans changer son PoW.

(3) Le niveau de sécurité n'est pas déterminé par le même aspect

- La sécurité économique est liée à Proof-of-Work / Proof-of-Stake
- La sécurité logique (accord, finalité) est liée à l'algorithme de consensus

Historique

Ces concepts ont des historiques différents

- Proof-of-Work vient de la cryptographie/anti-spam (années 90)
- Proof-of-Stake vient de la technologie blockchain (années 2010)
- Les consensus BFT viennent des systèmes distribués (années 80–90)

ETHEREUM

Origines

Créée en 2014 par Vitalik Buterin

Ethereum : A next-generation smart contract and decentralized 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 et un mécanisme PoW, l'idée de départ était de passer à une autre technologie 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. Voici une traduction.

"Je me demande si, à mesure que les bitcoins deviennent plus largement distribués, une transition d'un système basé sur la preuve de travail (PoW) vers un système basé sur la preuve d'enjeu (PoS) pourrait se produire. Ce que j'entends par PoS, c'est qu'au lieu que votre "vote" sur l'historique des transactions accepté soit pondéré par la part de ressources informatiques que vous apportez au réseau, il soit pondéré par le nombre de bitcoins dont vous pouvez prouver la possession grâce à vos clés privées."

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

Consensus 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 réutiliser 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

Motivations

Le problème du consensus distribué est un problème fondamental en Informatique

Le consensus distribué est un mécanisme permettant à un ensemble de participants indépendants de s'accorder sur une même valeur malgré les défaillances ou comportements malveillants

Applications du consensus distribué

Cryptomonnaies et blockchains : pour s'assurer que tous les participants s'accordent sur le même bloc, pour empêcher le double-spending.

Bases de données réparties : tous les serveurs doivent converger sur les mêmes données pour que les lectures soient cohérentes.

Serveurs cloud : pour élire un leader ou coordonner des actions sans conflit. Par exemple, choisir quel serveur gère une tâche critique dans un cluster Kubernetes.

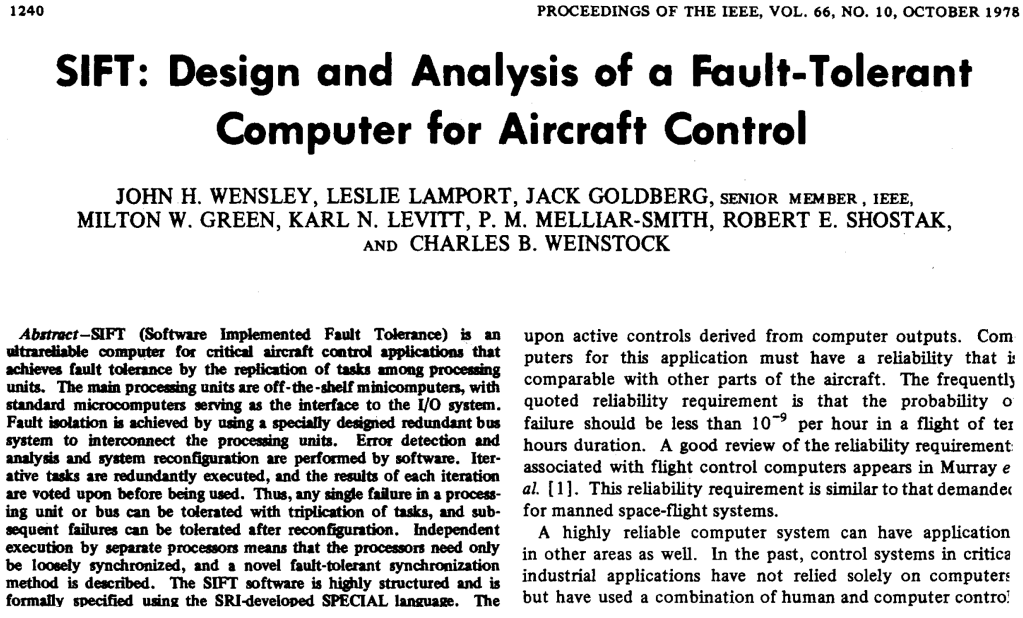
Systèmes critiques : plusieurs ordinateurs doivent s'accorder sur une décision même si certains capteurs ou calculateurs échouent.

IoT : coordonner des actions entre capteurs ou appareils décentralisés.

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.

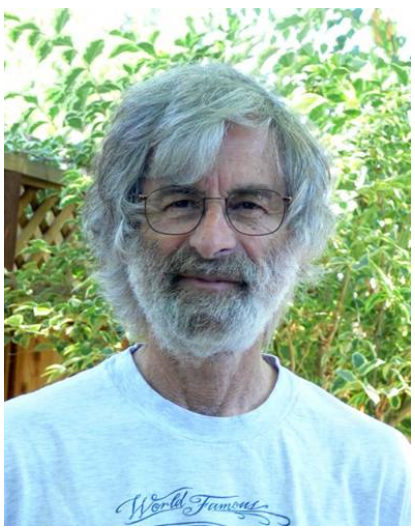
A highly reliable computer system can have application in other areas as well. In the past, control systems in critica industrial applications have not relied solely on computers but have used a combination of human and computer contro

- 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.

Hypothèses clés pour le consensus distribué

La faisabilité et l'efficacité d'un protocole de consensus dépendent fortement de certaines hypothèses

Réseau : fermé ou ouvert, complet (tous les participants peuvent communiquer directement) ou partiellement connecté

Communications : point-à-point ou broadcast, synchrones, partiellement synchrones ou asynchrones

Participants : fiables (honnêtes) ou byzantins (malveillants), avec ou sans pannes

Sécurité : confidentialité, authentification, intégrité des messages, infrastructure à clés publiques (PKI)

Hypothèses dans les blockchains

Les protocoles de consensus blockchain doivent gérer un environnement ouvert et peu fiable :

- **Fautes Byzantines** : certains participants peuvent être en panne, envoyer des messages arbitraires ou ne pas suivre le protocole.
- **Système ouvert** : tout le monde peut rejoindre, participants inconnus.
- **Pas d'information globale** : chaque participant n'a qu'une vision partielle du réseau.
- **Réseau P2P** : messages peuvent se perdre ou arriver en retard, temps de propagation non borné.

$\hookrightarrow$ Consensus : difficile, voire impossible à atteindre dans le cas général

CONSENSUS DISTRIBUÉ SUR RÉSEAU SYNCHRONE AVEC BYZANTINS

Communications synchrones

Les participants 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.

De manière équivalente, cette hypothèse permet de découper le temps en rounds, d'une durée égale à Δ et les participants 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.

Le problème du consensus distribué

On suppose un réseau de N participants ayant la forme d'un graphe de communication complet et non dirigé, avec des communications synchrones et point-à-point

Soit U un ensemble fixe et fini de valeurs.

Chaque participant a initialement une valeur initiale $V_i \in U$ ($i \in 1..N$)

Les participants souhaitent se mettre d'accord sur une même valeur $V \in U$. Pour cela, ils vont exécuter un protocole de consensus

Protocole de consensus

Un protocole de consensus peut se décomposer en trois étapes : start, rounds et end

> start: each participant i has an initial value V_i
> round(s) :
> each participant runs the same algorithm π, which can be divided into several rounds
> end: each participant outputs a value R_i

Un protocole de consensus doit satisfaire les propriétés suivantes :

- Agreement (ou consistency) : si deux participants i et j renvoient respectivement les valeurs R_i et R_j, alors $R_i = R_j$
- Termination : tous les participants finissent inévitablement par renvoyer une valeur de sortie.
- Validity : si tous les participants ont la même entrée V, alors ils renvoient tous V en sortie

Une première solution

Le problème du consensus a une solution très simple en un seul round

Chaque participant envoie sa valeur initiale à tous les autres, et lorsqu'un participant a reçu toutes les valeurs, il décide (à la fin du round), par exemple, d'une valeur maximale (ou tout autre critère)

Consensus avec pannes

Le problème du consensus devient intéressant (et difficile) lorsque n'importe quel participant peut tomber en panne : c'est-à-dire s'arrêter à tout moment (mais pas au milieu de l'envoi d'un message)

Selon les modèles, on peut considérer des pannes définitives ou temporaires, observables ou non, (automatiquement) détectable ou non par les autres processus

Consensus avec pannes (formalisation)

Il faut adapter les définitions des propriétés attendues pour une algorithme de consensus avec pannes

start: each participant i has an initial value V_i
round(s) :
    each participant runs the same algorithm π, which can be divided into several rounds
end: each non–crashed participant outputs a value R_i

Un protocole de consensus avec pannes doit satisfaire les propriétés suivantes :

- Agreement (ou consistency) : si deux participants qui ne sont pas tombés en panne i et j renvoient respectivement les valeurs R_i et R_j, alors $R_i = R_j$
- Termination : tous les participants qui ne sont pas tombés en panne finissent inévitablement par renvoyer une valeur de sortie.
- Validity : si tous les participants qui ne sont pas tombés en panne ont la même entrée V, alors ils renvoient tous V en sortie

Exercice 1

Donner un exemple d'exécution qui montre que la solution simple en un round ne fonctionne pas si un (ou plusieurs) participants tombent (définitivement) en panne

Trace de contre-exemple

On prend 4 participants tel que $A = 0, B = 0, C = 0, D = 1$

Round 1 :

- Les processus A, B et C envoient leur valeur initiale aux trois autres.
- Le processus D commence à envoyer sa valeur 1 :
  - $D \to A$: le message est envoyé et reçu dans le round.
  - Juste après cet envoi, D **tombe en panne**.
  - Les envois prévus vers B et C n'ont donc pas lieu.

Fin du round 1 :

- A a reçu les valeurs : $\{0_A, 0_B, 0_C, 1_D\}$.
- B a reçu : $\{0_A, 0_B, 0_C\}$.
- C a reçu : $\{0_A, 0_B, 0_C\}$.

Décisions :

$$\text{décision}(A) = 1, \text{décision}(B) = 0, \text{décision}(C) = 0$$

FloodSet

Une solution au problème du consensus distribué avec pannes a été publiée en 1983 par deux chercheurs de chez IBM, D. Dolev et H.R. Strong

SIAM J. COMPUT.
Vol. 12, No. 4, November 1983

0097-5397/83/1204-0005 $01.25/0

AUTHENTICATED ALGORITHMS FOR BYZANTINE AGREEMENT*

D. DOLEV† AND H. R. STRONG†

Abstract. Reaching agreement in a distributed system in the presence of faulty processors is a central issue for reliable computer systems. Using an authentication protocol, one can limit the undetected behavior of faulty processors to a simple failure to relay messages to all intended targets. In this paper we show that, in spite of such an ability to limit faulty behavior, and *no matter what message types or protocols* are allowed, reaching (Byzantine) agreement requires at least $t+1$ phases or rounds of information exchange, where t is an upper bound on the number of faulty processors. We present algorithms for reaching agreement based on authentication that require a total number of messages sent by correctly operating processors that is polynomial in both t and the number of processors, n. The best algorithm uses only $t+1$ phases and $O(nt)$ messages.

Key words. authentication, reliable distributed systems, Byzantine agreement, consistency, unanimity

1. Introduction. In this paper we consider algorithms for achieving agreement among multiple processors. The context for this agreement is a network of unreliable processors that have a means for conducting several synchronized phases of information exchange, after which they must all agree on some set of information. We will assume for simplicity that this set of information consists of a single value from some set of values V.

L'algorithme FloodSet

On suppose un ensemble de N participants, dont $f < N$ peuvent tomber en panne

L'idée principale de l'algorithme FloodSet est d'effectuer un nombre de rounds égal au nombre f de pannes que nous souhaitons tolérer, plus un round pour la décision finale.

round 0: each participant i maintains a variable W_i initialized to the singleton set $\{V_i\}$

round(s) $1 \leq k \leq f + 1$: each participant sends W_i to every other participant and adds all received values to W_i

`Decision rule` (at the end of round f+1):
For each participant i
if the set W_i contains only a single value v
then i decides v
else i decides a default value v_0

Exercice 2

Donner un exemple qui montre l'exécution de FloodSet pour 4 participants, dont 2 tombent en panne.

Trace pour 4 participants, 2 pannes

État initial	$W_1 = \{A\}, W_2 = \{B\}, W_3 = \{C\}, W_4 = \{D\}$
Round 1	P1 envoie à P2,P3,P4 P2 envoie à P1,P3,P4 P3 envoie à P1,P2 puis **crash** (n'envoie pas à P4) P4 envoie à P1 puis **crash** (n'envoie pas à P2,P3) **Réceptions :** P1 reçoit A,B,C,D $\Rightarrow W_1 = \{A, B, C, D\}$ P2 reçoit A,B,C $\Rightarrow W_2 = \{A, B, C\}$ P3 crash P4 crash
Round 2	Seuls P1,P2 envoient P1 $\rightarrow$ P2 : $\{A, B, C, D\}$ P2 $\rightarrow$ P1 : $\{A, B, C\}$ **Mises à jour :** $W_1 = \{A, B, C, D\}$; $W_2 = \{A, B, C, D\}$
Round 3	P1 $\leftrightarrow$ P2 (ensembles identiques) Pas de changement
État final	$W_1 = W_2 = \{A, B, C, D\}$; P3,P4 crashés

Autre trace (4 participants, 2 pannes)

État initial	$W_1 = \{A\}$, $W_2 = \{B\}$, $W_3 = \{C\}$, $W_4 = \{D\}$
Round 1	P4 envoie seulement à P1 puis **crash** **Réceptions :** P1 reçoit A,B,C,D $\Rightarrow W_1 = \{A, B, C, D\}$ P2 reçoit A,B,C $\Rightarrow W_2 = \{A, B, C\}$ P3 reçoit A,B,C $\Rightarrow W_3 = \{A, B, C\}$ P4 crash
Round 2	P1 envoie à P2 seulement puis **crash** P2 envoie à P1 et P3 **Réceptions :** P1 crash P2 reçoit D en plus $\Rightarrow W_2 = \{A, B, C, D\}$ P3 ne reçoit rien de nouveau $\Rightarrow W_3 = \{A, B, C\}$
Round 3	P2 $\leftrightarrow$ P3 P2 envoie $\{A, B, C, D\}$; P3 envoie $\{A, B, C\}$. W_3 devient $\{A, B, C, D\}$. Aucun autre changement.
État final	$W_2 = W_3 = \{A, B, C, D\}$; P1,P4 crashés.

Correction de l'algorithme FloodSet

Nous devons faire la preuve que FloodSet est correct, c'est-à-dire qu'il respecte les trois propriétés Agreement, Validity et Termination

On note $W_i(r)$ la valeur de W_i au round r

La preuve de correction de FloodSet repose sur les trois lemmes suivants :

Lemme 1 : Si aucun participant ne tombe en panne durant le round $1 \leq r \leq f+1$, alors $W_i(r) = W_j(r)$ pour tous les participants i et j non défaillants après r

Lemme 2 : Supposons que $W_i(r) = W_j(r)$ pour tous les participants i, j non défaillants après le round r. Alors, pour tout r' tel que $r \leq r' \leq f+1$, il en est de même, c'est-à-dire

$$W_i(r') = W_j(r') \quad \forall i, j \text{ non défaillants après le round } r'.$$

Lemme 3 : Si les participants i et j sont non défaillants après $f+1$ rounds, alors $W_i(f+1) = W_j(f+1)$

Preuve du lemme 1

Lemme 1 : Si aucun participant ne tombe en panne durant le round $1 \leq r \leq f + 1$, alors $W_i(r) = W_j(r)$ pour tous les participants i et j non défaillants après r

Preuve

Supposons qu'aucun participant ne tombe en panne durant le round r, et soit I l'ensemble des processus non défaillants après le round $r - 1$ (et donc r par hypothèse).

Comme chaque participant de I (et uniquement ceux-ci) envoie W à tous les autres processus au round r, par construction de l'algorithme, à la fin du round r,

$$W_i(r) = \bigcup_{j \in I} W_j(r-1).$$

Preuve du lemme 2

Lemme 2 : Supposons que $W_i(r) = W_j(r)$ pour tous les participants i, j non défaillants après le round r. Alors, pour tout r' tel que $r \leq r' \leq f + 1$, il en est de même, c'est-à-dire

$$W_i(r') = W_j(r') \quad \forall i, j \text{ non défaillants après le round } r'.$$

Preuve

On utilise une induction sur le nombre de rounds.

Le cas de base $r' = r$ correspond à l'hypothèse du lemme.

Pour le pas d'induction, il suffit de remarquer que tous les participants diffusent le même ensemble, et par conséquent aucun changement n'est apporté à W.

Preuve du lemme 3

Lemme 3 : Si les participants i et j sont non défaillants après $f+1$ rounds, alors $W_i(f+1) = W_j(f+1)$

Preuve

Comme il y a au plus f pannes, il existe un round r durant lequel aucune panne ne se produit.

Alors, le lemme 1 implique que

$$W_i(r) = W_j(r), \quad \forall i, j \text{ non défaillants après le round } r,$$

et le lemme 2 implique que

$$W_i(f+1) = W_j(f+1), \quad \forall i, j \text{ non défaillants après le round } f+1.$$

Preuve de l'algorithme FloodSet

Théorème : L'algorithme Floodset a les propriétés Agreement, Validity et Termination

Preuve

- **Termination :** découle trivialement du modèle synchrone des communications et de l'absence de boucles (ou d'attentes)
- **Validity :** si tous les participants ont la même valeur initiale v, alors tout message transmis contiendra toujours $\{v\}$, donc

$$W_i(r) = \{v\} \quad \forall i \text{ non défaillant après } r \leq f + 1 \text{ rounds.}$$

  Cela implique que la valeur décidée sera v.
- **Agreement :** comme tous les participants décident après $f + 1$ rounds, le lemme 3 implique que tout couple de processus qui décident ont la même valeur pour W.

Consensus distribué avec Byzantins

On s'intéresse maintenant à une extension de l'algorithme FloodSet en présence de participants Byzantins

Pannes (ou fautes) Byzantines : certains participants peuvent être en panne, envoyer des messages arbitraires ou ne pas suivre le protocole

Le problème des généraux Byzantins

Dans LSP'82, le problème du consensus Byzantin est illustré de la manière suivante :

The Byzantine Generals Problem

LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE
SRI International

Reliable computer systems must handle malfunctioning components that give conflicting information to different parts of the system. This situation can be expressed abstractly in terms of a group of generals of the Byzantine army camped with their troops around an enemy city. Communicating only by messenger, the generals must agree upon a common battle plan. However, one or more of them may be traitors who will try to confuse the others. The problem is to find an algorithm to ensure that the loyal generals will reach agreement. It is shown that, using only oral messages, this problem is solvable if and only if more than two-thirds of the generals are loyal; so a single traitor can confound two loyal generals. With unforgeable written messages, the problem is solvable for any number of generals and possible traitors. Applications of the solutions to reliable computer systems are then discussed.

Categories and Subject Descriptors: C.2.4. [**Computer-Communication Networks**]: Distributed Systems—*network operating systems*; D.4.4 [**Operating Systems**]: Communications Management—*network communication*; D.4.5 [**Operating Systems**]: Reliability—*fault tolerance*

General Terms: Algorithms, Reliability

Additional Key Words and Phrases: Interactive consistency

Consensus avec Byzantins (formalisation)

Il faut adapter les définitions des propriétés attendues pour une algorithme de consensus avec Byzantins

start: each participant i has an initial value V_i
round(s) :
    each participant runs the same algorithm π, which can be divided into several rounds
end: each honest participant outputs a value R_i

Un protocole de consensus avec Byzantins doit satisfaire les propriétés suivantes :

- Agreement (ou consistency) : si deux participants honnêtes i et j renvoient respectivement les valeurs R_i et R_j, alors $R_i = R_j$
- Termination : tous les participants honnêtes finissent inévitablement par renvoyer une valeur de sortie.
- Validity : si tous les participants honnêtes ont la même entrée V, alors ils renvoient tous V en sortie

Le problème de la diffusion Byzantine

On peut restreindre le problème des généraux Byzantins à celui de la diffusion Byzantine (Byzantine Broadcast – BB) : un général envoie un ordre à ses $N - 1$ lieutenants.

Le général ou les lieutenants peuvent être Byzantins.

On souhaite avoir les deux propriétés suivantes pour un protocole implémentant une diffusion Byzantine :

- Consistency : Tous les lieutenants honnêtes doivent obéir au même ordre.
- Validity : Si le général est honnête, alors tous les lieutenants honnêtes reçoivent (et obéissent à) son ordre.

L'algorithme de Dolev-Strong (FloodSet avec Byzantins)

N participants, dont $f < N$ peuvent être Byzantins

L'algorithme de Dolev-Strong est basé sur la même idée que FloodSet : on va effectuer $f + 1$ rounds

En plus de l'hypohèse de synchronisme faite pour FloodSet, on va utiliser une infrastructure d'authentification PKI

Ainsi, on note `sign(`V`,` i`)` pour indiquer que la valeur V est signée par la participant i

Dolev-Strong

Chaque participant i possède une variable W_i qui va contenir un ensemble de messages signés.

Le leader est supposé être le participant numéro 1.

round 0: The leader (participant 1) sends $\langle V_1, \texttt{sign}(V_1, 1)\rangle$ to every participant

round(s) $1 \leq k \leq f + 1$:
- if participant i receives for the first time value v signed by k distinct participants
- then i adds v to W_i and sends $\langle v, \texttt{sign}(v, i)\rangle$ to all participants

Decision rule (at the end of round f+1):
- if W_i contains only a single value v
- then participant i outputs v
- else i outputs the default value v_0

Exercice 4

Donner un exemple qui montre l'exécution de Dolev-Strong pour 4 participants, dont 2 Byzantins

Correction de Dolev-Strong

Dolev-Strong est une implémentation correcte de BB

Validity :

- Si le Leader est honnête, tous les participants honnêtes reçoivent la valeur V_1 envoyée par le Leader au round 1er round
- Grâce aux signatures numériques, les participants honnêtes ne peuvent recevoir une autre valeur que celle envoyée par le Leader.

Agreement :

- Si un message contient $f + 1$ signatures, alors cela signifie qu'un participant honnête l'a signé.
- Si un participant honnête a signé un message, alors tous les participants honnêtes vont le recevoir dans le round qui suit.

Consensus pour réseaux synchrones

Dans les réseaux synchrones, on distingue deux grandes familles d'algorithmes de consensus :

- **Catégorie 1** : Algorithmes itératifs avec $f + 1$ rounds
- **Catégorie 2** : Algorithmes par quorums avec 1 round

Pour classer ces algorithmes, on distinguera également :

- Nature des pannes (crash, omission, byzantine)
- Présence ou absence de signatures (authentifié / non-authentifié)

Catégorie 1 : Algorithmes itératifs avec $f + 1$ rounds

Ces algorithmes fonctionnent en propageant de manière répétée des valeurs reçues.

- On itère suffisamment longtemps pour “purger” les comportements fautifs.
- Le nombre minimal de rounds dépend du type de panne et des signatures.

Les rounds servent à compenser un nombre maximum de f pannes. Avec (au moins) $f + 1$ rounds, on est certain qu'un round s'est déroulé sans pannes.

$\hookrightarrow$ La construction de ces algorithmes itératifs, et leur preuve de correction, reposent sur ce fait.

Catégorie 1 : modèles de pannes

Pannes par crash/omission

FloodSet : s'il n'y a aucune panne durant un round, alors tous les participants honnêtes se retrouvent avec le même ensemble de valeurs, et cet ensemble restera identique pour tous jusqu'à la fin de l'exécution.

Pannes byzantines non authentifiées

Oral Messages : si, dans une étape récursive, aucun message n'est falsifié, alors tous les participants honnêtes construisent exactement la même partie de l'arbre du message. Cette partie restera identique pour tous et conduira finalement à la même valeur décidée.

Pannes byzantines authentifiées

Dolev-Strong : si un message a $f + 1$ signatures distinctes, alors au moins un signataire est honnête. Comme un honnête diffuse toujours les messages qu'il signe, tous les autres honnêtes reçoivent forcément cette information. C'est ce qui garantit que les participants honnêtes partagent la même connaissance après $f + 1$ rounds.

Catégorie 2 : Algorithmes par quorums

Les algorithmes par quorums reposent sur un unique round qui sert à capturer l'opinion (les votes) des participants.

Afin de dégager une super-majorité honnête, ces algorithmes reposent sur la récolte de deux quorums Q_1 et Q_2 de même cardinalité $N - f$.

La correction de l'algorithme de consensus repose toujours sur le nombre de votes honnêtes que l'on peut déduire de la cardinalité de l'intersection $|Q_1 \cap Q_2|$ qui est donnée par :

$$\begin{aligned} |Q_1 \cup Q_2| = |Q_1| + |Q_2| - |Q_1 \cap Q_2| &\leq N \\ N - f + N - f - |Q_1 \cap Q_2| &\leq N \\ |Q_1 \cap Q_2| &\geq N - 2f \end{aligned}$$

Catégorie 2 : modèles de pannes

Pannes par crash/omission

Pour garantir la sûreté d'un algorithme de consensus avec des pannes de type crash ou omission, il faut être sûr d'avoir au moins un participant dans $Q_1 \cap Q_2$ qui a participé correctement aux deux quorums au moment du vote.

Pour cela, il faut donc que $|Q_1 \cap Q_2| \geq N - 2f = 1$

C'est possible si le nombre de participants est $N = 2f + 1$.

Pannes byzantines

De même, pour garantir la sûreté du consensus avec des pannes byzantines, il faut être sûr d'avoir au moins $f + 1$ votes dans $Q_1 \cap Q_2$. Cela permet de garantir qu'il existe un participant honnête commun à ces deux quorums.

Pour cela, il faut donc que $|Q_1 \cap Q_2| \geq N - 2f = f + 1$

C'est possible si le nombre de participants est $N = 3f + 1$.

CONSENSUS AVEC QUORUMS

Streamlet

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

STREAMLET: Textbook Streamlined Blockchains

Benjamin Y Chan[1] and Elaine Shi[2]

[1]Cornell University - byc@cs.cornell.edu
[2]CMU/Cornell - runting@gmail.com

September 12, 2020

Abstract

In the past five years or so, numerous blockchain projects have made tremendous progress towards improving permissioned consensus protocols (partly due to their promised applications in Proof-of-Stake cryptocurrencies). Although a significant leap has silently taken place in our understanding of consensus protocols, it is rather difficult to navigate this body of work, and knowledge of the new techniques appears scattered.

In this paper, we describe an extremely simple and natural paradigm called STREAMLET for constructing consensus protocols. Our protocols are inspired by the core techniques that have been uncovered in the past five years of work; but to the best of our knowledge our embodiment is simpler than ever before and we accomplish this by taking a "streamlining" idea to its full potential. We hope that our textbook constructions will help to decipher the past five years of work on consensus partly driven by the cryptocurrency community — in particular, how remarkably simple the new generation of consensus protocols has become in comparison with classical mainstream approaches such as PBFT and Paxos.

Streamlet : hypothèses

- 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 ≠ 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.

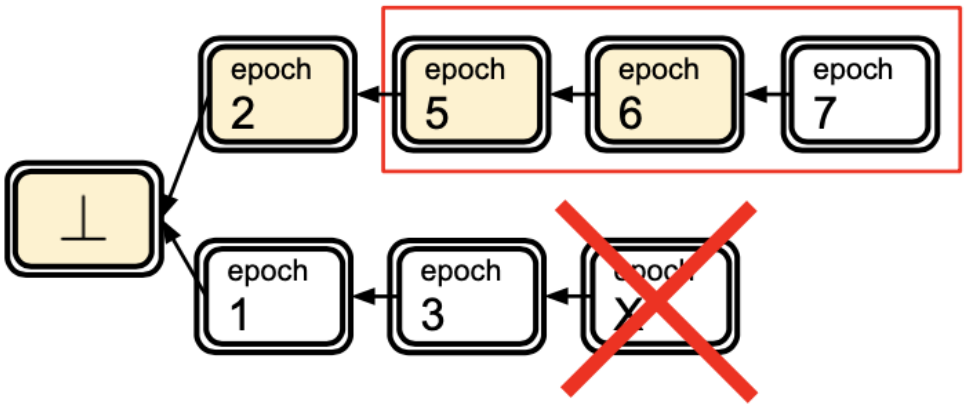
Un vote est simplement un message contenant la signature de n pour le bloc b.

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 $\square$.

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 $\square$.

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 $ch'[:l'-1]$.

RÉSULTATS D’IMPOSSIBILITÉ

Rappels

Byzantine Broadcast (BB) :

- Un nœud unique joue le rôle de leader.
- Chaque nœud honnête termine (termination).
- Tous les nœuds honnêtes renvoient la même valeur (agreement).
- Si le leader est honnête, chaque nœud honnête envoie la valeur du leader (validity)

Byzantine Agreement (BA) :

- Chaque nœud a sa propre valeur en entrée.
- Agreement, Termination identiques à BB.
- Validity : si tous les nœuds honnêtes ont la même valeur b en entrée, alors ils renvoient tous cette valeur b.

Dans les slides suivants, on utilise ”consensus” pour BA ou BB.

Résultat d'impossibilité dans les réseaux synchrones

Sous les hypothèses suivantes :

- communications synchrones,
- algorithme déterministe,
- pas de PKI
- protocole déterministe
- jusqu'à f byzantins (pour un total de N participants)

$\hookrightarrow$ Le problème du consensus (BB, BA) est impossible si $f \geq N/3$

Ce résultat est dû à Pease, Shostak et Lamport (PSL80), et une preuve plus simple a été donnée par Fischer, Lynch et Merritt (FLM86).

Synchronisme ?

Rappel :

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 :

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)

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 $f \geq N/3$