Daniele Piscaglia

Come rendere Hyperledger Fabric una blockchain

Cos'è Hyperledger Fabric

Hyperledger Fabric (HLF) si propone come una soluzione blockchain permissioned, in grado di gestire un ledger distribuito in cui è coinvolto un numero di organizzazioni che si trovano tutte allo stesso livello gerarchico e decisionale, senza che sia richiesto alcun rapporto di fiducia tra di esse. In HLF ognuna di queste organizzazioni viene registrata e gestita tramite un sistema di autenticazione che permette di limitare gli accessi alle sole identità "conosciute".

Panoramica di Hyperledger Fabric

Come funziona Hyperledger Fabric

In un tipico deployment di HLF sono presenti 4 principali figure:

- Client: gli effettivi utilizzatori della blockchain che richiedono operazioni e transazioni tramite l'esecuzione di sequenze di codice remoto chiamate chaincode
- Peer: coloro che eseguono i chaincode e verificano le transazioni dei client
- Orderer: può essere una o più figure che si occupano di dare un ordine temporale deterministico alle transazioni che vengono aggiunte alla blockchain
- CA (Certificate Authority): registra e gestisce tutti i certificati necessari all'autenticazione (solitamente una per ogni organizzazione)

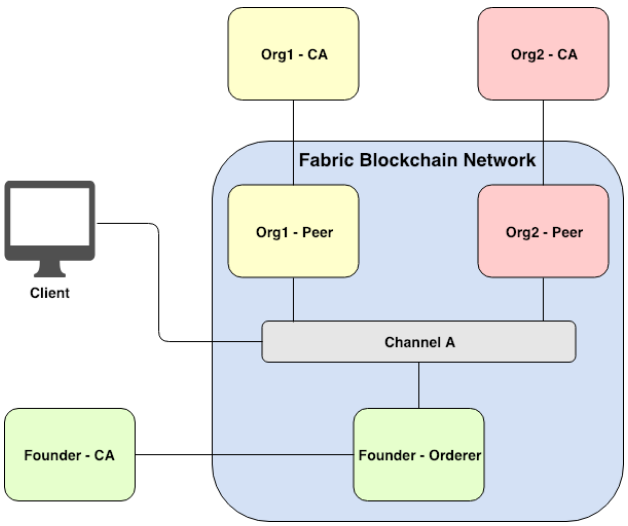

Figura 1 Architettura di Hyperledger Fabric

Tutte le comunicazioni tra le parti passano attraverso un channel, se si desidera separare le comunicazioni e nascondere alcune operazioni di ambiti diversi è anche possibile definire più channel. In un utilizzo reale di HLF, si può immaginare la partecipazione di più organizzazioni, che assieme contribuiscono alla creazione della rete HLF ma allo stesso tempo non hanno nessuna garanzia di fiducia reciproca, dove idealmente ogni organizzazione partecipa alla rete inserendo un proprio peer ed un proprio orderer.

Ogni volta che in HLF un client deve eseguire un chaincode[1] invia una richiesta ad uno o più peer e si entra così nella fase chiamata "**Endorsement Phase**". Durante questa fase in base alla policy di sicurezza che è stata definita i peer verificano la transazione richiesta dal client e inviano una notifica di conferma (risposte firmate crittograficamente dal peer) al client. Quando il client ha ricevuto abbastanza "conferme" le inoltra ad uno degli orderer.
A questo punto comincia la "**Ordering Phase**", nel caso l'ordering service sia composto da un solo nodo (cioè nella rete ci sia un solo orderer) è evidente che l'ordine di ricezione delle richieste dai client coincida con l'ordine dell'inserimento delle transazioni nei blocchi. Nel caso invece siano presenti più orderer questa fase richiede una coordinazione globale in modo da accordarsi univocamente e globalmente su quale sia l'ordine finale per inserire le transazioni nei blocchi.

Le problematiche

HLF è stato sin dalla nascita un sistema dalla natura configurabile, permettendo la scelta da una vasta lista di implementazioni e configurazioni diverse per ognuna delle sue parti che si è continuamente evoluta con l'avanzamento del progetto. La grande possibilità di personalizzazione ha portato l'interesse di molti sviluppatori e ricercatori a effettuare numerosi benchmark con lo scopo di confrontare i vantaggi di ogni possibile soluzione, test che però solitamente si concentravano sulla parte di endorsement, focalizzandosi quindi sul linguaggio di programmazione dei chaincode, endorsement policies, gestione dei peer ecc. mentre invece viene spesso ignorato l'aspetto di ordering in HLF.

In realtà però, l'ordinamento è una fase importantissima ed è proprio il meccanismo di sincronizzazione distribuito che copre il ruolo di meccanismo di consenso in HLF. Mentre la parte di endorsement (che assicura che le transazioni vengano eseguite in maniera corretta, senza violazioni di permessi) esegue in maniera completamente distribuita (e decentralizzata) da parte di tutte le organizzazioni partecipanti, quella di ordering è ancora centralizzata (eseguita da un solo nodo) per garantire maggiore velocità, dove gli altri nodi (rappresentanti le altre organizzazioni) sono solamente delle copie di backup che prendono il posto del leader in caso di errore. Questo rende il sistema CFT (Crash Fault Tolerant, resistente ai guasti) ma certamente non BFT (Byzantine Fault Tolerant).

Cos'è un sistema BFT

In un sistema, oltre al normale crash di un nodo, che semplicemente smette di funzionare o rispondere (ed è quindi necessario intervenire per garantire il funzionamento del sistema) può essere anche previsto un "Byzantine Crash" (BC). Nel caso di un BC, oltre agli errori standard si prevede anche che il nodo stesso possa essere in realtà malevolo, quindi non solo smetta di inviare i messaggi o eseguire operazioni in maniera corretta ma possa addirittura cominciare a operare in modo incorretto e compromettente sulla rete (favorendo ad esempio una delle parti). Questo è ovviamente uno scenario importante da considerare in un sistema come HLF, dove come abbiamo già detto si trovano a partecipare organizzazioni che non hanno alcun rapporto di fiducia e quindi in qualsiasi momento una organizzazione (o più di esse in accordo) possa cominciare ad operare secondo i propri interessi. Se un sistema è in grado di contrastare che questo succeda e quindi anche in uno scenario simile continua a funziona come previsto, allora il sistema può essere definito BFT (Byzantine Fault Tolerant). Esistono da tempo meccanismi di protezione da queste minacce, che permettono di garantire la sicurezza in base alla percentuale del numero di nodi

[1] Sostituiti degli usuali Smart Contract in HLF, dei veri e propri programmi che vengono invocati dai client.

intrusori rispetto a quella dei nodi onesti che però come vedremo introducono spesso problemi di scalabilità.

Perché Fabric dovrebbe essere BFT

Per quanto riguarda HLF in realtà, è un sistema che è nato come sistema BFT, grazie a una delle implementazioni più basilari, la Practical Byzantine Fault Tolerance, che garantiva sicurezza fino ad una percentuale di nodi onesti circa del 33% (*f* nodi intrusi su un totale di *3*f +1* nodi) ma con importanti problemi di scalabilità sul numero di orderer che hanno costretto gli sviluppatori ad abbandonare l'implementazione sin dalla versione 0.6 (Gennaio 2017), sacrificando quindi la capacità BFT della blockchain [1]. Attualmente quindi sono implementati altri due meccanismi per la fase di ordinamento (più uno per il solo scopo di sviluppo e test) che sono chiamati Kafka e l'ultimo arrivato Raft, in realtà le differenze a livello di funzionalità tra i due sono minime, ma Raft ha reso molto più facilmente mantenibile e configurabile tutta la fase di deployment e ottimizzato le performance diventando quindi la scelta standard. Il problema di entrambi i sistemi però risulta essere la loro natura CFT, cioè non fanno altro che occuparsi della sincronizzazione e replica del nodo orderer che sta attualmente ricoprendo il ruolo di leader e quindi garantiscono il corretto funzionamento anche nel caso in cui questo cessi di funzionare, facendo subentrare una replica. È evidente che quindi la fase di ordinamento non è in alcun modo decentralizzata in quanto sotto il completo controllo dell'organizzazione che ricopre il ruolo di leader.

Questa decisione porta ad un lista di potenziali minacce che non sono facilmente ignorabili in una blockchain, infatti se supponiamo il leader dell'ordinamento compromesso, per quest'ultimo diventa possibile controllare a proprio piacimento quali transazioni vengono aggiunte al ledger per prime (potendo quindi dare la precedenza a membri specifici rendendo il sistema non equo) e soprattutto impedire completamente la pubblicazione di transazioni di un certo tipo o di un certo utente (censurando quindi la blockchain). Anche se queste minacce possono risultare un compromesso accettabile per molti utilizzi, rendono HLF un sistema che in realtà non aggiunge funzionalità o sicurezze che lo rendono molto differente rispetto ad un tradizionale database centralizzato che sarà sicuramente in grado di fornire performance più elevate ad un costo inferiore. Per rendere HLF realmente interessante per un utilizzo concreto si vuole quindi fare un analisi di una possibile soluzione, che senza avere un impatto troppo elevato sulle performance, introduca quelle proprietà di controllo decentralizzato che i database tradizionali non possono offrire.

Il problema è stato affrontato più volte, tra oppositori di HLF, i quali richiedono una completa resistenza BFT prima di poterla considerare una blockchain, mentre i sostenitori di HLF ritengono che sia del tutto superflua, data la natura permissioned che richiede una "validazione" di tutte le parti prima di poter entrare a far parte della rete. Quindi si propone di seguito uno scenario concreto di utilizzo di HLF per la creazione di un ecosistema che gestisca la compravendita di un qualsiasi tipo di asset. In uno scenario centralizzato si prevedrebbe che tutti i clienti che vogliono effettuare una compravendita lo facciano tramite un ente centrale che quindi ordina e registra le varie transazioni, supponiamo quindi che l'ente che ricopre il ruolo di leader per l'ordinamento faccia anche parte della rete come utente (direttamente o indirettamente tramite un utente "sybil[2]").

[2] Con la parola sybil si intendono tutte quelle identità digitali che anche se non apparentemente collegate vengono controllate dalla stessa entità fisica.

Ora supponiamo uno scenario in cui si hanno due utenti onesti (un venditore e un acquirente) intenti ad effettuare lo scambio di un qualsiasi asset ad un determinato prezzo. Per il controllore dell'ordinamento compromesso risulta facile "oscurare" la compravendita tra i due utenti legittimi oscurandoli a vicenda (bloccando le loro transazioni) e sostituire il suo utente "sybil" come acquirente. A questo punto, in seconda battuta, in quanto nuovo possessore dell'asset a cui l'acquirente originale era interessato, può procedere alla vendita ad un prezzo da lui stabilito. E' evidente che se quell'asset risulta ora disponibile solo presso il venditore fraudolento (o in alternativa le altre offerte di vendita legittime sono state oscurate), l'acquirente sarà obbligato ad acquistarlo al nuovo prezzo deciso dall'intermediario.

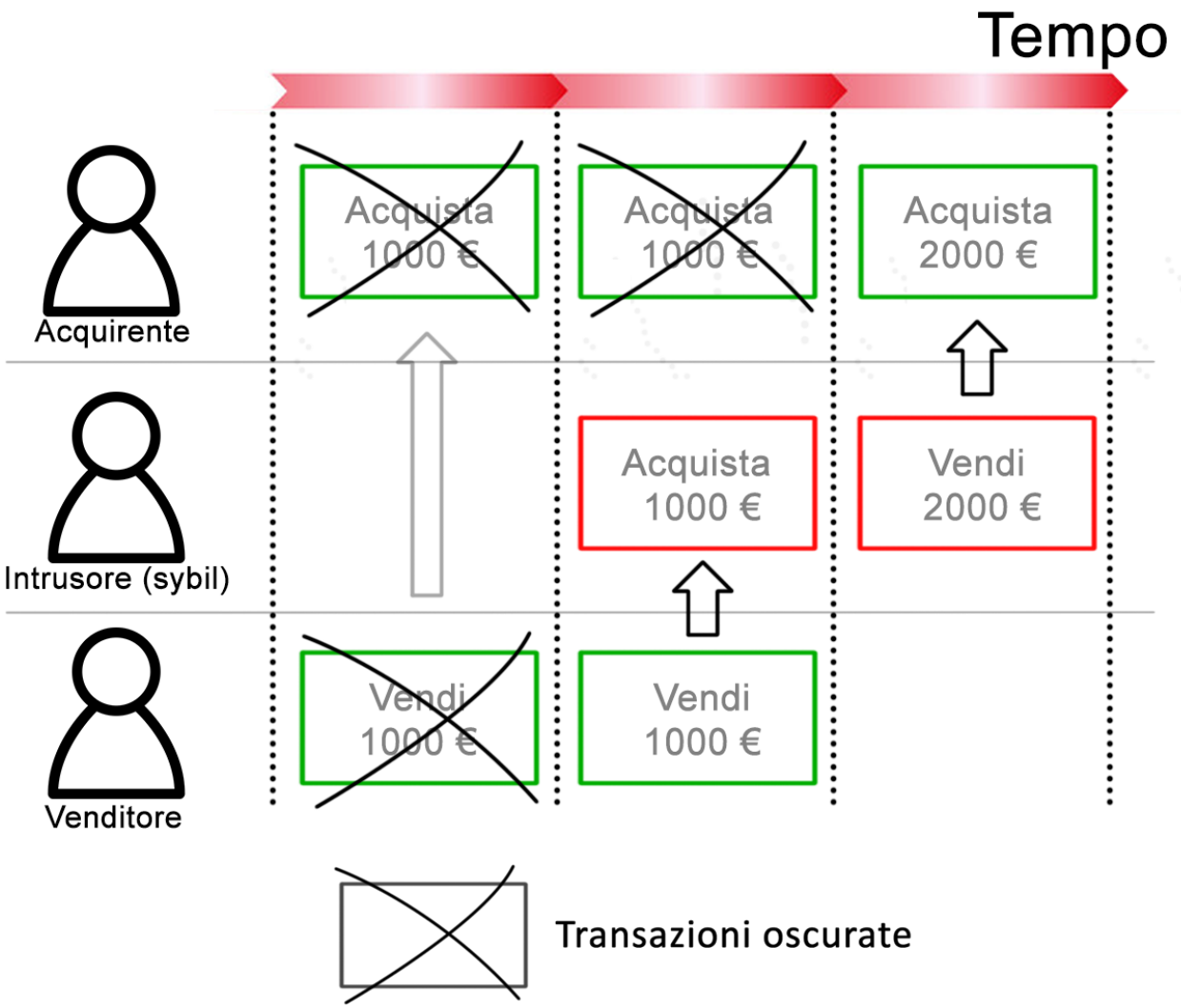

Figura 2 Esempio di intromissione da parte del leader

Questo semplice esempio dimostra come nel caso di un sistema ad ordinamento non BFT sia richiesta l'assoluta fiducia da parte degli utilizzatori verso chi controlla l'ordinamento. Nel caso invece, di un sistema in cui anche l'ordinamento è decentralizzato e BFT nessuna parte può avere potere decisionale unilaterale (come ad esempio la possibilità di favorire la priorità di alcune transazioni o di ignorarne altre), senza che sia quindi richiesto alcun tipo di fiducia verso nessuna delle parti.

Proof of Majority

Algoritmo che viene presentato [2] come una nuova soluzione molto efficiente, dove i nodi anziché competere in una elezione per stabilire il nuovo generatore di un blocco, collaborano contemporaneamente producendo lo stesso blocco. La produzione concorrente dello stesso blocco presso più nodi, formandone più copie è il processo stesso di conferma, che si considera completato quando il numero di copie è tale da raggiungere la maggioranza nella rete.

A questo scopo, per ogni blocco vengono definite le seguenti proprietà (incluse nell'header):

- Numero di transazioni contenute nel blocco
- Peso (*weight*) del blocco
- Conferma (*finalized*) del blocco

Ogni nodo partecipante al processo, riceve tutte le transazioni che devono essere aggiunte alla blockchain e quindi procede alla creazione del blocco stesso in maniera completamente autonoma. Ogni nuovo blocco viene creato con il valore di peso (*w*) a 1 e la conferma (*f*) a false, che indica che il blocco non è ancora da considerarsi "valido". Questo processo viene ripetuto da tutti i nodi sulla rete, portando quindi (esclusi eventuali problemi di partizionamento) alla creazione di numerosi blocchi identici. Successivamente, tutti i blocchi con lo stesso hash (gemelli) possono quindi essere uniti (merge) in un unico blocco che però avrà un peso (*w*) pari al numero di blocchi identici che lo "confermano" e potrà quindi essere finalizzato (campo *finalized* con valore true).

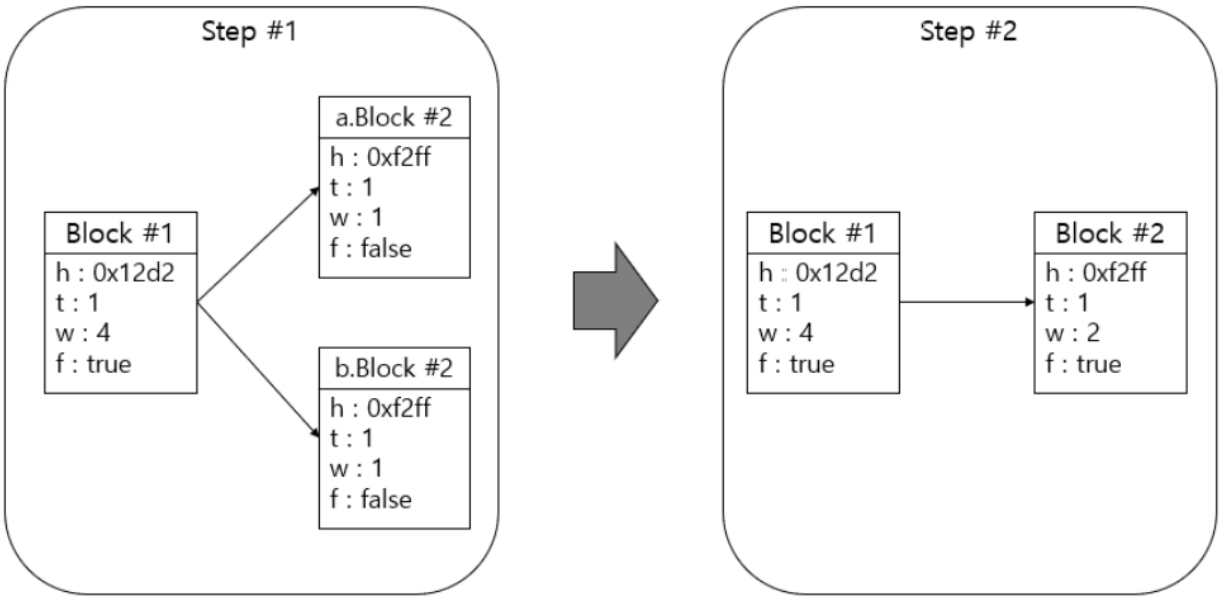

Figura 3 Processo di creazione e conferma di un blocco

È tutta via possibile che per qualche motivo, nella rete si verifichi un fork[3] nel quale vengono creati contemporaneamente due blocchi contrastanti (vedi figura sotto), in tal caso quindi, dopo il merge tra le varie versioni in competizione verrà selezionata quella con il peso più alto e solamente quella verrà confermata impostando il valore finalized a true mentre l'altra verrà ignorata.

[3] Biforcazione, contrasto di due "versioni" contrastanti sulla blockchain, cioè due blocchi diversi che dovrebbero occupare la stessa posizione

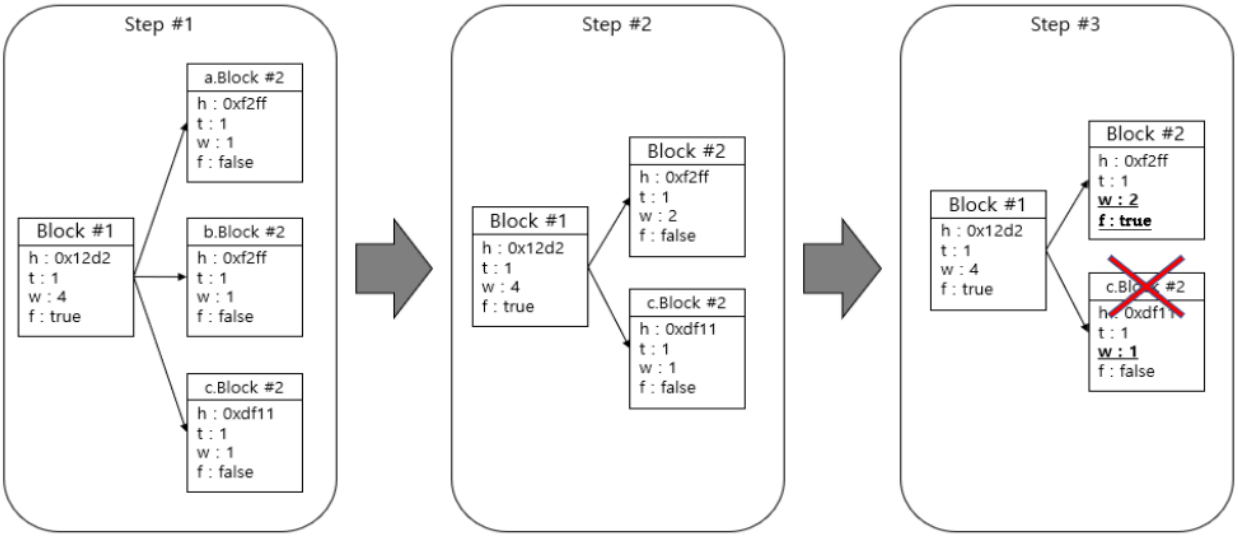

Figura 4 Processo di convergenza dopo un fork

Questo meccanismo di consenso, si integra perfettamente in un ambiente permissioned, sfruttando la conoscenza delle identità dei nodi per evitare attacchi di tipo Sibyl[4], è infatti molto importante che nessuna organizzazione abbia la possibilità di controllare più nodi partecipanti, riconosciuti con identità differenti con lo scopo di avere a disposizione più "voti" e quindi poter controllare la maggioranza.

Tutto il processo rimane estremamente semplice, senza la necessità di sfruttare calcoli complessi per garantire la sicurezza, che risulta assicurata fino a quando il numero di nodi onesti supera il numero di nodi disonesti; inoltre non è necessario alcuno scambio di messaggi diretto per la sincronizzazione tra i nodi (come nelle fasi di pre-prepare, prepare e commit della practical byzantine fault tolerance). Proprio per questo motivo l'algoritmo PoM appena descritto si presenta come il candidato ottimale per svolgere il compito di ordinamento all'interno di Fabric permettendo una scalabilità del numero di orderer significativamente migliore rispetto alle soluzioni già viste.

Proof of Majority in Hyperledger Fabric

In base a quanto visto di HLF possiamo affidare alla fase di endorsement il processo di verifica dell'esecuzione di un determinato chaincode, assicurando che le modifiche che esso apporta al ledger non portino ad inconsistenze oppure coinvolgano dati sui quali non si è autorizzati ad operare, grazie alle svariate politiche di endorsement già definibili nella piattaforma.

Ora prendiamo però in considerazione i vantaggi che l'algoritmo Proof of Majority (PoM) porterebbe ad HLF se utilizzato durante la fase di ordinamento, passando quindi dal paradigma precedente di master e replica degli orderer a un nuovo schema completamente paritario in cui tutte le organizzazioni partecipanti svolgono un ruolo attivo. Mentre nello schema precedente era sufficiente inviare la transazione al master orderer, che la applicava e poi aggiornava la stato delle repliche, per il funzionamento del PoM è necessario che tutti i nodi partecipanti (in questo caso gli orderer) siano a conoscenza delle transazioni da applicare. Questo problema può essere risolto

[4] Scenario in cui un attaccante guadagna la maggioranza sulla rete prendendo il controllo di più nodi apparentemente distaccati e non collegati tra loro

inviando dal client la nuova richiesta a tutti gli orderer, oppure incaricando l'orderer che per primo riceve la richiesta di un client di informare tutti gli altri.

Mentre entrambe le soluzioni sono possibili, nel secondo caso si evita di caricare di responsabilità il client che potrebbe ad esempio non è essere a conoscenza della completa lista degl'orderer, inoltre per la comunicazione si potrebbe utilizzare un canale di broadcast specifico tra gli orderer che sarebbe comunque necessario realizzare per il trasferimento dei nuovi blocchi creati.

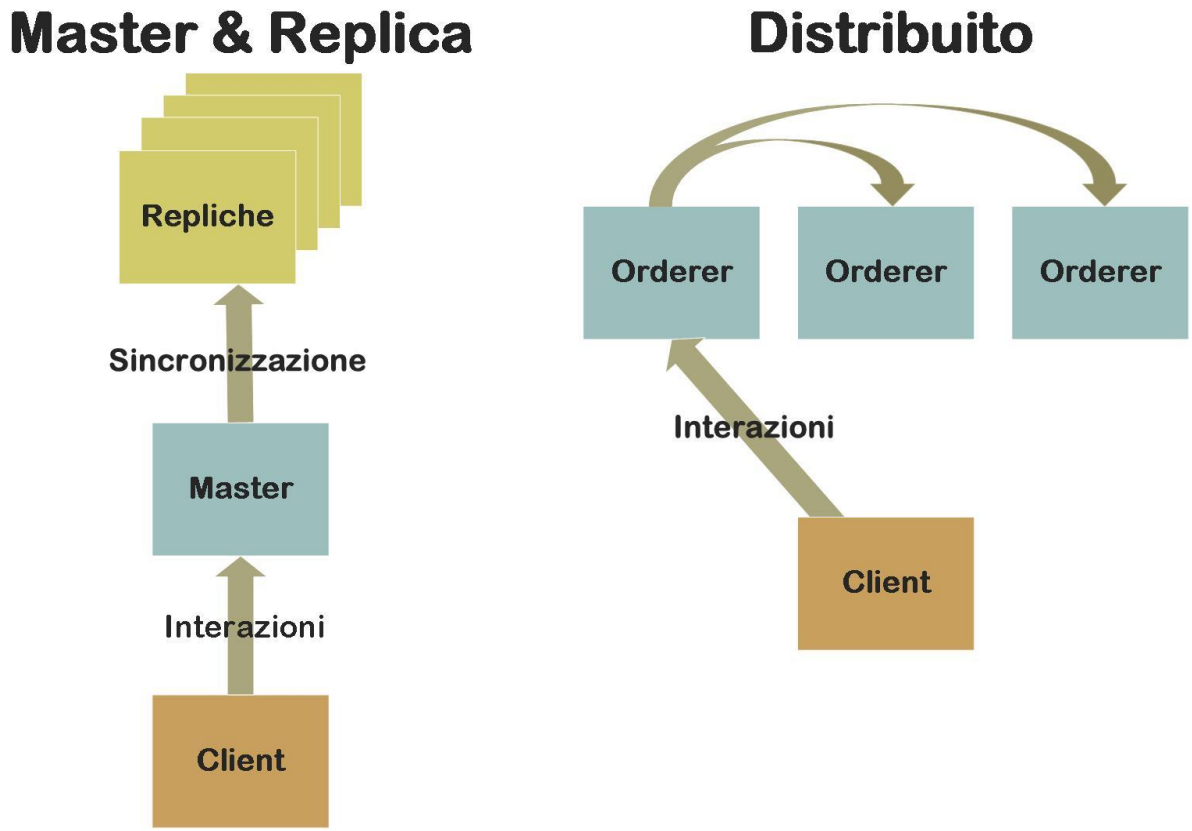

Figura 5 Confronto gestione orderer (attuale vs proposta)

Per quanto riguarda lo stato attuale di HLF la dimensione dei blocchi nei quali le transazioni vengono raggruppate è limitato in base ad alcuni parametri di dimensione e di numero di transazioni per ogni blocco. Per l'utilizzo dell'algoritmo PoM risulta però conveniente organizzarle in finestre temporali, in base al timestamp delle transazioni, che può essere applicato al momento della ricezione dal primo orderer (prima di inoltrare la richiesta agl'altri) in modo da essere univoco su tutta la rete. Questo meccanismo permette al sistema di compensare la latenza di comunicazione tra i vari nodi evitando il sorgere di conflitti nell'ordinamento di transazioni temporalmente "vicine".

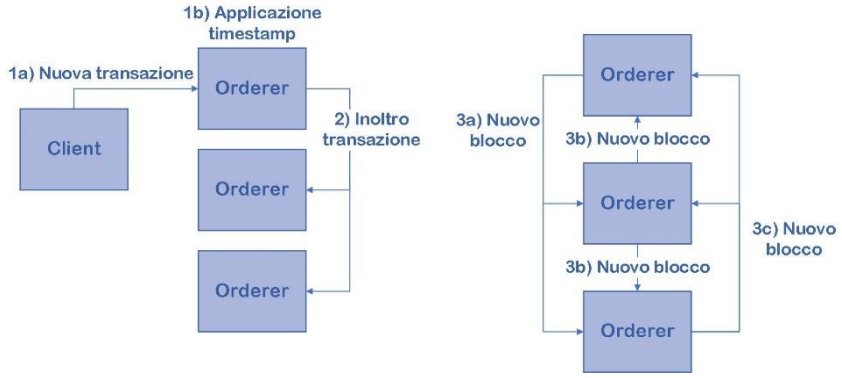

Figura 6 Ordine interazioni per ogni nuova transazione (a sinistra) e per ogni nuovo blocco (a destra)

Tutti i nodi quindi procederanno alla creazione dei blocchi ad intervalli prefissati (ad esempio ogni 30 secondi) inserendovi le transazioni (ricevute dai client e/o dagli altri orderer) dotate di un timestamp che rientra nella finestra temporale prevista.

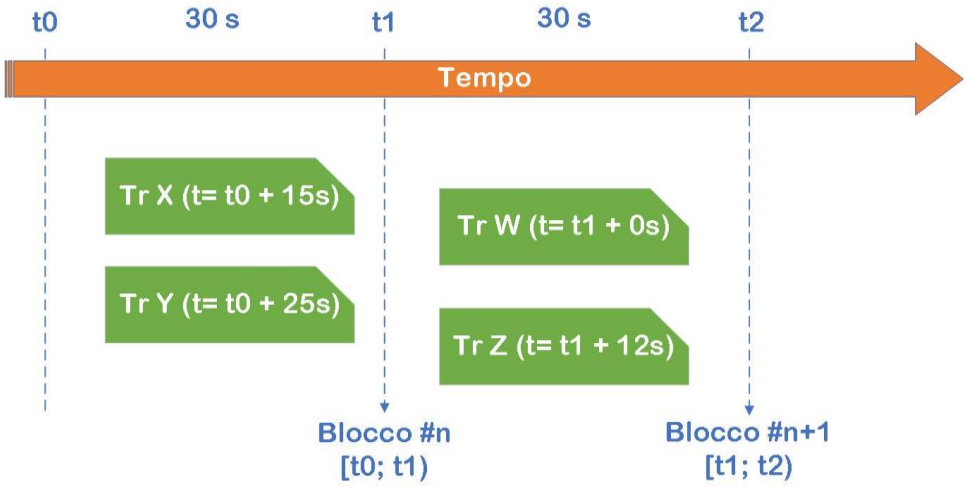

Figura 7 Esempio di collocazione temporale delle transazioni all'interno dei nuovi blocchi

L'utilizzo di queste finestre temporali permette di compensare probabili disallineamenti degli orologi dei vari nodi e dei ritardi dovuti ai trasferimenti, collocando all'interno dello stesso blocco le stesse transazioni presso tutti i nodi della rete. Per quanto riguarda la gestione del ritardo di propagazione della rete[5] si può pensare che ogni nodo, al termine di una finestra, anziché procedere immediatamente alla creazione del blocco aspetti una quantità di tempo tale da permettere alle transazioni ancora in rete (non ancora ricevute) di arrivare a destinazione e quindi procedere alla creazione di un blocco allineato al resto della rete. Come in gran parte dei sistemi distribuiti, non si può presupporre che il carico e la velocità di arrivo di nuove transazioni sia costante nel tempo, quindi si avranno sicuramente blocchi con un numero di transazioni (e quindi una dimensione) variabile. Per consentire una diffusione "abbastanza veloce" del blocco sul resto della rete si può anche pensare di inserire un limite sulla dimensione, nel caso arrivi un numero di transazioni tali da superare tale limite, le più nuove (con il timestamp più recente) saranno quindi messe in attesa (in una sorta di coda di backlog) e rimandate al blocco successivo. Idealmente per

[5] Che anche nel caso di reti globali rimane comunque sotto i 400 ms, soglia oltre la quale l'infrastruttura TCP/IP stessa diventa inutilizzabile

non avere ritardi troppo alti è preferibile che la coda di attesa sia vuota, ma in casi estremi permette di gestire scenari di sovraccarico temporanei.

Il passo di validazione finale dei blocchi prodotti può quindi avvenire in autonomia presso ognuno dei nodi, che una volta ricevuta dagl'altri partecipanti una quantità di copie di un blocco tale da raggiungere la maggioranza sulla rete può considerare il blocco stesso valido. E' importante sottolineare quanto questa fase sia efficiente (in quanto non richiede nessuna ulteriore comunicazione tra i nodi) e univoca, infatti anche presso un nodo che per qualsiasi problema di partizionamento si è prodotto un blocco differente dal resto della rete (blocco X), alla ricezione di una quantità di maggioranza di copie di un'altra versione (blocco Y), viene preferita la visione condivisa dalla rete rispetto a quella locale anche presso il nodo disallineato (il blocco X viene annullato e il blocco Y validato).

Per quanto riguarda la consultazione del ledger rimangono aperte svariate possibilità, infatti fino adesso abbiamo considerato un canale dove solo gli orderer vengono informati sulla creazione di nuovi blocchi, ma in realtà si può pensare che chiunque possa mettersi in ascolto (mentre la pubblicazione deve essere concessa solo agli orderer specificati), permettendo quindi agli endorsing peer di mantenere in locale una copia sempre aggiornata del ledger (come succede nella attuale implementazione di HLF) o persino la creazione di ipotetici "heavy client"[6].

Risulta quindi un ruolo piuttosto chiave quello ricoperto dal canale di broadcast per il trasferimento dei blocchi (affiancato da quello per l'inoltro delle nuove transazioni), per il quale è richiesto un alto livello di affidabilità e velocità. Il problema però è di dominio piuttosto ampio ed è già stato affrontato più volte in passato nel campo dei sistemi distribuiti, portando alla creazione di numerosi sistemi ed implementazioni per la comunicazione di tipologia publish-subscribe[7] chiamati MQTT[8]. Si può infatti pensare a una soluzione dove sono definiti due topic (nuove transazioni e nuovi blocchi) sui quali i vari nodi si sottoscriveranno (solo gli orderer al primo e tutti gli aventi una copia della blockchain al secondo). Grazie all'utilizzo di questi sistemi già disponibili sul mercato quindi si può avere una soluzione molto efficiente, affidabile e scalabile (come dimostrato da svariate analisi, come in quella svolta da ScaleAgent [3]).

E' importante sottolineare anche come la natura e l'architettura di HLF non vengano in realtà stravolte dall'introduzione dell'algoritmo PoM, che continua a permettere il normale funzionamento di tutte le parti (al di fuori degli orderer che vengono sostanzialmente ricostruiti) senza il bisogno di grandi modifiche, soprattutto per quanto riguarda le funzionalità offerte. Uno degli aspetti più importanti è quello della *finalità* della rete (finality in inglese), cioè il fatto che le transazioni, una volta validate e pubblicate sulla blockchain non possano essere più annullate o ripristinate, cioè che non ci saranno mai delle biforcazioni (forks) sulla rete. Bisogna però fare una distinzione tra le transazioni pubblicate e quelle validate, infatti similmente a quanto accade nello stato attuale di HLF, è perfettamente possibile che una determinata transazione (quindi l'esecuzione di un chaincode) abbia superato la policy di endorsement, si stata inserita in un blocco dagli orderer ma in realtà quando viene pubblicata sulla blockchain si trova in una posizione

[6] Heavy client sono considerati quei client che mantengono in locale una copia completa del ledger in modo da poterlo consultare senza aver bisogno di contattare nessun altro nodo (al contrario dei light client)
[7] Sistemi di messaggistica dove chi vuole ricevere (chiamati subscriber) si sottoscrive ad un determinato topic e riceverà un qualsiasi messaggio quando viene pubblicato su tale topic da un qualsiasi publisher
[8] Message Queue Telemetry Transport

tale da operare su dei dati obsoleti o non validi (ad esempio se un'altra transazione, pubblicata prima di lei ha già modificato quei dati). In questo caso la transazione verrà ovviamente considerata non valida da tutti i partecipanti della rete, quindi in quanto tale è come se non appartenesse affatto alla blockchain (le sue modifiche non verranno considerate). La probabilità che un evento simile accada è sicuramente legata alla scrittura del chaincode, che deve quindi essere organizzato in modo da utilizzare il meno possibile dati in maniera concorrente, ma anche alla finestra di tempo utilizzata per la creazione di blocchi. Una finestra molto grande porterebbe più transazioni all'interno dello stesso blocco (quindi "concorrenti") e implicherebbe un ritardo maggiore tra il momento in cui la transazione viene validata dagli endorser e quando viene effettivamente pubblicata, aumentando quindi le probabilità che essa venga validata considerando dati obsoleti. Allo stesso tempo però, una finestra molto piccola richiederebbe ai nodi di essere in ottima sincronia tra di loro in modo da non incorrere in disallineamenti nella fase di creazione dei blocchi. Quindi questo parametro andrà sicuramente configurato con attenzione nello specifico ambiente finale, tenendo in considerazione i ritardi che la rete dovrà gestire.

Related Works

Nonostante il team di sviluppo principale di Fabric abbia rimandato la implementazione di un ordinamento BFT, altre comunità di sviluppatori si sono impegnate a fornire delle alternative e mentre alcune non abbiano mai superato lo stadio di semplice proposta teorica, altre sono arrivate fino ad avere un implementazione funzionante perfettamente integrata con Fabric. L'implementazione più apprezzata è senz'altro quella presentata come l'adattamento di una libreria esistente chiamata BFT-Smart [4], nonostante la compatibilità sia limitata solamente ad alcune versioni della release principale della blockchain. Un altro esempio è la soluzione proposta con il nome di RCanopus [5], ad ogni modo il problema principale che tutte queste soluzioni proposte condividono, risulta il fatto che siano state ideate sulla base dell'algoritmo della Practical Byzantine Fault Tolerance (pBFT)[6]. Algoritmo che per la sincronizzazione prevede lo scambio di un elevato numero di messaggi che cresce esponenzialmente al crescere dei nodi partecipanti. Le varie soluzioni approcciano il problema con soluzioni e ottimizzazioni diverse, per cercare di risolvere il problema, ma come risulta piuttosto evidente nel caso di BFT-Smart (l'unico attualmente implementato) e dai test effettuati è complicato sostenere scenari con un numero di orderer che va oltre la decina.

Grazie alla soluzione proposta con Proof Of Majority invece si riescono a introdurre due principali vantaggi. Innanzitutto si riduce il numero dei messaggi che sono necessari per la sincronizzazione dei nodi ad ogni "consensus round" (cioè alla creazione di ogni blocco), nel caso sia disponibile un qualsiasi canale broadcast per la comunicazione verso gli altri nodi richiedendo un solo messaggio inviato da ogni nodo per ogni ciclo. Una quantità inferiore di messaggi (nella figura un confronto tra i numeri di messaggi unicast) permette di evitare il congestionamento della rete durante le fasi di validazione con conseguente perdita di pacchetti.

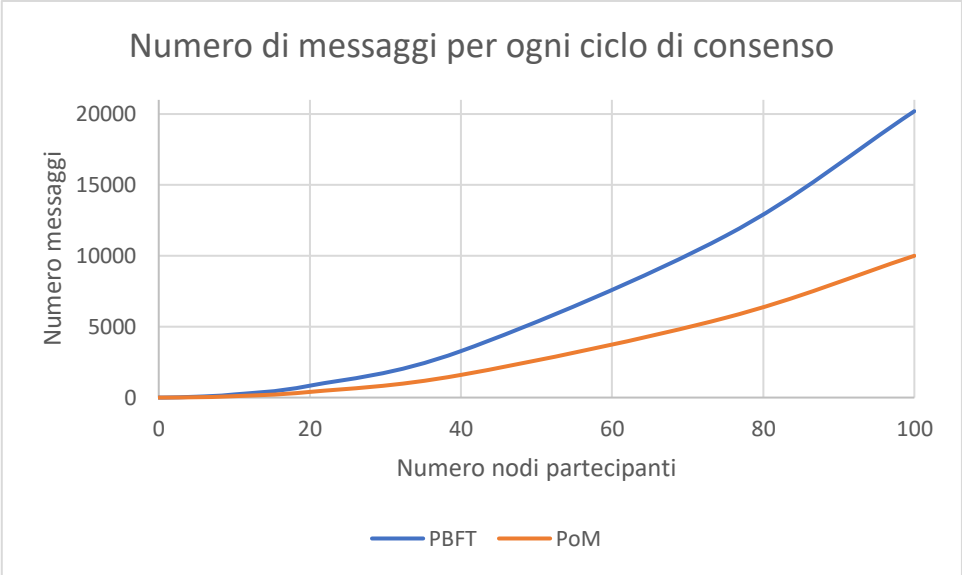

Figura 8 Confronto tra i messaggi in pBFT e PoM

Il secondo maggiore vantaggio è la possibilità di effettuare la sincronizzazione dei partecipanti in una singola fase. Volendo fare un confronto con l'algoritmo pBFT, dove erano necessarie 3 fasi (pre-prepare, prepare e commit), le prime due fasi vengono eliminate, risparmiando quindi ad ogni nodo le attese legate alle latenze dei messaggi che dovevano essere ricevuti in successione e riducendo i ritardi dovuti alla trasmissione fino ad un terzo del tempo. Nella nuova proposta PoM infatti le interazioni tra i nodi si possono considerare "scollegate" tra loro in quanto non è mai previsto che all'invio di un qualsiasi messaggio sia collegata l'attesa di una qualsiasi risposta o l'innesco di una reazione dalla controparte, rendendo la natura dell'algoritmo completamente asincrona (e quindi particolarmente appropriata ad un sistema distribuito).

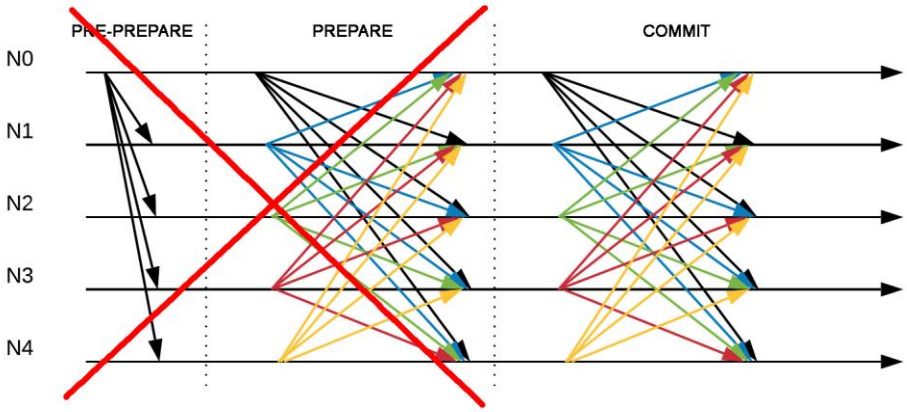

Figura 9 Confronto tra pBFT e le fasi di sincronizzazione risparmiate in PoM

Conclusioni

Con le considerazioni fatte in questo documento quindi, si vuole comunicare quanto sia facile inserire un vero e proprio algoritmo di consenso distribuito all'interno di Hyperledger Fabric grazie alla sua natura componibile e configurabile, soprattutto si vuole anche sottolineare perché Proof Of Majority sia particolarmente adatto a tale scopo, fornendo una soluzione ottima e completamente nuova, che si adatta alle caratteristiche attuali della blockchain. Ad ogni modo si sono anche affrontate alcune domande e possibili problematiche che potrebbero presentarsi nel caso di un utilizzo reale.

Un aspetto fondamentale per il funzionamento e la sicurezza dell'algoritmo PoM è il numero di nodi partecipanti, si può quindi considerare l'algoritmo utilizzabile solo a partire da una determinata soglia minima del numero di nodi, per questo motivo Proof Of Majority non si propone come una alternativa completamente distruttiva, ma come una nuova possibilità che potrebbe integrarsi con la attuale implementazione, utilizzandola come meccanismo di avvio fino al raggiungimento delle condizioni necessarie per poi effettuare una "conversione" a PoM. Si è anche voluto fare un confronto rispetto alla pBFT, in quanto unica alternativa realmente BFT che sia mai stata implementata in Fabric, evidenziandone i vantaggi introdotti in termini di efficienza mantenendo lo stesso grado di sicurezza. Non avendo attualmente una implementazione tangibile dell'algoritmo non è possibile definire in termini chiari le performance che può garantire, ad ogni modo, per le considerazioni viste sembrerebbe essere una delle soluzioni più veloci ed efficaci per affrontare il problema della Byzantine Fault Tolerance all'interno di Hyperledger Fabric.

Riferimenti

[1] IBM, «Fabric Ordering decentralization,» [Online]. Available: https://developer.ibm.com/articles/blockchain-hyperledger-fabric-ordering-decentralization/.

[2] J.-T. Kim, J. Jin e K. Kim, «A study on an energy-effective and secure consensus algorithm for private blockchain systems,» [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8539561/.

[3] ScaleAgent, «MQTT services benchmark,» 2015. [Online]. Available: http://www.scalagent.com/IMG/pdf/Benchmark_MQTT_servers-v1-1.pdf.

[4] J. Sousa, A. Bessani e M. Vukolic, «A BFT Ordering Service for Fabric,» 2017. [Online]. Available: https://arxiv.org/pdf/1709.06921.pdf.

[5] S. Keshav, W. Golab, B. Wong, S. Rizvi e S. Gorbunov, «RCanopus: Making Canopus Resilient to Failures and Byzantine Faults,» [Online]. Available: https://www.researchgate.net/publication/328446093_RCanopus_Making_Canopus_Resilient_to_Failures_and_Byzantine_Faults.

[6] M. Castro e B. Liskov, «Practical Byzantine Fault Tolerance,» 1999. [Online]. Available: http://pmg.csail.mit.edu/papers/osdi99.pdf.