ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria e Architettura
Corso di Laurea in Ingegneria e Scienze Informatiche

Hyperledger: Architettura, Struttura e Tecnologie applicate alla Blockchain

Tesi di laurea in
INGEGNERIA DEL SOFTWARE

Relatore
Chiar.mo Prof. Andrea Omicini

Correlatore
Dott. Giovanni Ciatto

Candidato
Cristiano Aprigliano

Prima Sessione di Laurea
Anno Accademico 2020-2021

Sommario

La tecnologia della Blockchain porta diversi benefici per le aziende come immutabilità, trasparenza, tracciabilità e la possibilità di sviluppare la propria Blockchain mentre per gli utenti finali avranno il pieno controllo dei loro dati e *assets*. La Blockchain è un progetto Open-Source al quale chiunque può contribuire nello sviluppo. E' possibile implementarla e utilizzarla in diversi settori aziendali ed è per questo che l'interesse nei suoi confronti è in costante aumento.

Hyperledger è un progetto Open-Source gestito dalla Community dedicato allo sviluppo di soluzioni stabili per facilitare l'uso e l'implementazione della tecnologia della Blockchain per le aziende attraverso l'uso di Frameworks, strumenti e librerie. Al momento della stesura di questa tesi, Hyperledger conta sei Frameworks, quattro librerie e cinque strumenti (tools). Data l'architettura modulare dei Frameworks, ogni azienda può modellare ed adattare praticamente ogni aspetto del framework scelto ed implementarlo ad-hoc, per risolvere un determinato problema.

L'obiettivo di questa tesi è quello di illustrare il funzionamento, la struttura e l'architettura di una Blockchain e dei principali sei frameworks sviluppati da Hyperledger.

If you don't believe me or don't get it,
I don't have time to try to convince you, sorry.

Satoshi Nakamoto, *tratto da Bitcointalk.org*

Ringraziamenti

Ringrazio il mio relatore Andrea Omicini e il mio correlatore Giovanni Ciatto per avermi ispirato e assegnato il progetto. Infine, ringrazio i miei amici e la mia famiglia per avermi supportato durante l'intera durata di questo percorso.

Indice

Sommario **iii**

1 Introduzione **1**

2 Stato dell'Arte **3**
2.0.1 Caratteristiche di una Blockchain 4
2.1 Classificazione Blockchain 5
2.1.1 Blockchain Permissionless 5
2.1.2 Blockchain Permissioned 5
2.2 Componenti Blockchain 6
2.2.1 Funzioni Hash Crittografiche 6
2.2.2 Nonce Crittografico 8
2.2.3 Transazioni 8
2.2.4 Blocchi 10
2.2.5 Blocchi concatenati 11
2.2.6 Smart Contracts 12
2.2.7 Processi di uno Smart Contract 14
2.3 Consenso 15
2.3.1 Tipi di Consenso 15
2.3.2 Proprietà del Consenso 17

3 Hyperledger **19**
3.1 Architettura di Hyperledger 20
3.2 Progetti Hyperledger 22
3.3 Hyperledger Besu 24
3.3.1 Caratteristiche di Hyperledger Besu 24
3.4 Hyperledger Burrow 28
3.4.1 Caratteristiche di Hyperledger Burrow 28
3.5 Hyperledger Fabric 31
3.5.1 Smart Contracts e Caratteristiche di Hyperledger Fabric . . 32
3.5.2 Consenso in Hyperledger Fabric 33

3.6 Hyperledger Indy 35
3.6.1 Caratteristiche di Hyperledger Indy 35
3.6.2 Consenso in Hyperledger Indy 36
3.7 Hyperledger Iroha 38
3.7.1 Caratteristiche chiave di Hyperledger Iroha 38
3.7.2 Consenso in Hyperledger Iroha 38
3.8 Hyperledger Sawtooth 42
3.8.1 Caratteristiche di Hyperledger Sawtooth 42
3.8.2 Installed Smart Contracts con Transaction Families 43
3.8.3 On-Chain Smart Contracts con Seth Transaction Family . . 43
3.8.4 Consenso in Hyperledger Sawtooth 44

4 Conclusioni **47**

Bibliografia **50**

Elenco delle figure

2.1 Esempio di una Blockchain [1]. 4
2.2 Esempio di una Transazione di Criptomoneta [2]. 9
2.3 Concatenazione dei Blocchi 11
2.4 Illustrazione di uno Smart Contract 12
2.5 Come gli Smart Contract processano le richieste [3]. 14
2.6 Tabella Comparativa degli approcci di Consenso Permissioned e Standard Proof of Work [4]. 16
2.7 Processo di Consenso Generalizzato di Hyperledger [4]. 17

3.1 Filosofia di Design di Hyperledger [5]. 20
3.2 Ethereum Virtual Machine 24
3.3 Struttura Proof of Authority 25
3.4 Esempio del processo di una possibile transazione in Hyperledger Fabric [4]. 33
3.5 Rappresentazione di RBFT [4]. 36
3.6 Passaggi di un Protocollo RBFT [4]. 37
3.7 Flusso di una transazione in Hyperledger Iroha con Algoritmo Sumeragi [4]. 39
3.8 Flusso di una transazione in Hyperledger Iroha con Algoritmo Sumeragi nel caso di un Errore nel Server [4]. 40

Capitolo 1

Introduzione

Nonostante la Blockchain sia una tecnologia giovane, negli ultimi anni si è potuto osservare l'incremento di interesse relativo alle sue potenzialità. Sempre molte più aziende di vari settori ed industrie investono ed iniziano a sviluppare soluzioni ad-hoc per le loro necessità, basate sulla Blockchain.

Questa tendenza porta diversi benefici per le aziende come immutabilità, trasparenza, tracciabilità e la possibilità di sviluppare la propria Blockchain mentre per gli utenti finali avranno il pieno controllo dei loro dati e *assets*, non doversi fidare di servizi di terze parti per la convalida delle transazioni e dei contratti. Utilizzando un qualsiasi sevizio basato sulla Blockchain come l'invio di una semplice transazione o la stipulazione di un Contratto, ci si "fida" del codice e non di un'entità, che può essere ad esempio una Banca o un'Istituzione. La fiducia nel codice della Blockchain infatti è data dal fatto che è un Progetto Open-Source, nel quale chiunque può visionare il codice ed il suo completo funzionamento, in una singola parola, è *trasparente*.

Hyperledger è un progetto Open-Source gestito dalla Community dedicato allo sviluppo di soluzioni stabili per facilitare l'uso e l'implementazione della tecnologia della Blockchain per le aziende attraverso l'uso di frameworks, strumenti e librerie. Molte aziende, tra cui Walmart, hanno implementato uno di questi Framework, in questo caso *Hyperledger Fabric*, in uno dei loro processi più importanti: la filiera alimentare (food supply chain). Il problema era appunto relativo alla tracciabilità dei cibi in caso si sviluppasse un focolaio di una malattia di origine alimentare; il processo di tracciabilità per risalire a quale fornitore è dovuto necessitava di settimane, mentre dopo l'implementazione di *Fabric*, 2.2 secondi.

L'obiettivo di questa tesi non è soltanto mostrare il lato implementativo, ma fornire anche una dettagliata panoramica del funzionamento di una Blockchain e dei principali frameworks di Hyperledger.

Capitolo 2

Stato dell'Arte

La Blockchain può sembrare un concetto complicato, è quindi opportuno capire quale sia il suo concetto principale. Possiamo quindi asserire che la Blockchain è un tipo di Database. Un database è una collezione di informazioni che vengono immagazzinate elettronicamente su un sistema informatico. Le informazioni, o dati, sono tipicamente strutturati in tabelle per permettere una più semplice ricerca e filtraggio per informazioni specifiche.

Una differenza sostanziale tra un tipico Database ed una Blockchain è come sono strutturati i dati. La Blockchain raccoglie i dati in gruppi, chiamati Blocchi, i quali contengono set di informazioni. I blocchi hanno determinate capacità di memoria e quando sono pieni, vengono agganciati ai blocchi precedenti formando una catena di dati, da qui, Blockchain.

Ne deduciamo quindi che tutte le Blockchain sono Database, ma non tutti i Database sono Blockchain.

Questo sistema intrinsecamente comporta ad una irreversibile sequenza temporale di dati quando implementata in modo decentralizzato. Ad ogni blocco infatti, viene assegnato un esatto orario (timestamp) quando viene aggiunto alla catena.

Prendendo in considerazione la Blockchain di Bitcoin (2.1), i blocchi che vengono successivamente appesi alla Blockchain, contengono multiple transazioni, le quali andranno a formare un registro completo di tutte le transazioni avvenute fino a quel momento.

Oltre alle informazioni relative alle transazioni, ogni blocco contiene un *timestamp* (data e ora), l'*hash* del blocco precedente ("parent") ed un *nonce*, il quale è un numero random utile alla verifica dell'hash.

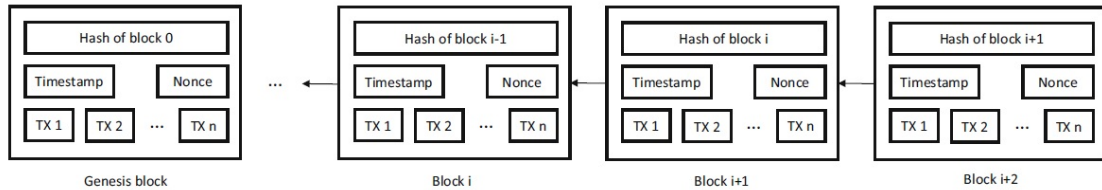

Figura 2.1: Esempio di una Blockchain [1].

Questa struttura dei blocchi garantisce quindi l'integrità di tutta la blockchain fino al primo blocco (chiamato *Genesis Block*).

I valori di Hash relativi ai blocchi sono unici e grazie a questo, si possono prevenire le frodi, in quanto il più piccolo cambiamento apportato ad un blocco, darebbe come risultato un valore di hash completamente diverso.

2.0.1 Caratteristiche di una Blockchain

- Registro (*Ledger*) - La tecnologia della Blockchain utilizza un approccio *append-only* (appendi alla fine) per fornire lo storico completo delle transazioni avvenute.
- Sicuro - Le Blockchains sono crittograficamente sicure, assicurando che i dati contenuti all'interno del registro non siano stati alterati e che i dati presenti all'interno del registro siano affidabili.
- Condivisa - Il registro è condiviso tra molteplici partecipanti. Questo fornisce trasparenza tra i partecipanti dei nodi nella rete della Blockchain.
- Distribuita - La Blockchain può essere distribuita. Questo permette l'incremento del numero dei nodi della rete blockchain per renderla più resistente a potenziali attacchi.

2.1 Classificazione Blockchain

Le reti Blockchain possono essere categorizzate in base al loro modello di autorizzazione, il quale determina chi può mantenerle (es. pubblicare blocchi). Se chiunque può pubblicare un nuovo blocco, la Blockchain viene chiamata *permissionless* (senza autorizzazioni). Se solo particolari utenti possono pubblicare nuovi blocchi, viene chiamata *permissioned* (con autorizzazioni).

2.1.1 Blockchain Permissionless

Le reti Blockchain *permissionless* sono registri decentralizzati aperti a chiunque voglia pubblicare un blocco, senza la necessità di avere i permessi necessary da alcuna autorità. Le piattaforme Blockchain permissionless sono spesso software open-source, liberamente disponibili a chiunque desideri scaricarle.

Dato che chiunque ha il privilegio di pubblicare un blocco, questo comporta che chiunque possa leggere ed effettuare una transazione sulla Blockchain. Conseguentemente, un utente malevolo potrebbe provare a pubblicare un blocco in modo da sovvertire il sistema. Per prevenire ciò, le Blockchain *permissionless* spesso utilizzano un accordo multiparte o sistema di '*consenso*' che richiede agli utenti di spendere o mantenere risorse per poter pubblicare nuovi blocchi. Alcuni esempi di modelli di *consenso* includono i metodi di *Proof of Work* e *Proof of Stake*.

2.1.2 Blockchain Permissioned

Le reti Blockchain *Permissioned* sono controllate da un'Autorità, spesso chi la sviluppa e mantiene, i quali gestiscono i permessi degli utenti, tra cui la possibilità di pubblicare nuovi blocchi.
Dato che solo gli utenti autorizzati manutengono la Blockchain, è possibile limitare l'accesso di lettura e limitare chi può effettuare transazioni. Le Blockchain *Permissioned* potrebbero quindi permettere a chiunque di leggere la Blockchain oppure limitare l'accesso di lettura ad individui autorizzati. Di conseguenza, potrebbero permettere a chiunque di effettuare transazioni da includere nella Blockchain, o ancora, potrebbero limitare questa funzionalità solo ad individui autorizzati.

Allo stesso modo delle reti Blockchain *Permissionless*, usano modelli di consenso per la pubblicazione di blocchi, ma in questo caso, spesso non richiedono una spesa o manutenzione delle risorse. Grazie al fatto che è necessario registrare l'identità dell'utente per partecipare come membro della rete, coloro che manutengono la Blockchain hanno un livello di fiducia tra di loro, dato che tutti sono

stati autorizzati a pubblicare i blocchi e potrebbe essergli rimossa l'autorizzazione in caso non si comportino correttamente.

Le organizzazioni che hanno bisogno di proteggere e controllare in modo più stretto la propria Blockchain, potrebbero avere necessità di un'implementazione *Permissioned*. Tuttavia, se una singola organizzazione o entità, ha il pieno controllo della Blockchain, sarà necessario che gli utenti abbiano fiducia in quella entità o organizzazione. Una Blockchain *Permissioned* è utile anche in caso due o più aziende debbano lavorare insieme ma non si fidano completamente l'uno dell'altra. E' quindi possibile stabilire una rete Blockchain *Permissioned* ed invitare le altre entità a registrare le loro transazioni su un registro distribuito condiviso.

2.2 Componenti Blockchain

La tecnologia Blockchain può sembrare complessa ma può essere semplificata esaminando ogni componente individualmente. Ad alto livello, la tecnologia Blockchain utilizza meccanismi informatici ben noti e primitive crittografiche come le Funzioni Hash Crittografiche, Firme Digitali e Crittografia a Chiave Asimmetrica, insieme a concetti di conservazione dei registri (come i registri (*ledgers*) *append-only*).

2.2.1 Funzioni Hash Crittografiche

Un importante componente della tecnologia Blockchain è l'uso delle Funzioni Hash Crittografiche per molte operazioni. Applicandole ai dati, esse calcolano e restituiscono un output relativamente unico (chiamato *message digest* o *digest*) per un input di qualsiasi dimensione e tipologia (es. file, testo, immagine). L'utilizzo di queste Funzioni permette ad ogni individuo di verificare l'autenticità dell'input, in quanto una qualsiasi modifica di quest'ultimo (anche in termini di bit), comporterebbe un output totalmente differente.

Le Funzioni Hash Crittografiche possiedono inoltre importanti proprietà relativamente alla sicurezza:

- **Resistenza alla preimmagine:** sono funzioni unidirezionali; è computazionalmente impossibile calcolare il valore di input a partire da uno specifico output (es. dato un *digest*, trovare x tale che $hash(x) = digest$).

- **Resistenza alla seconda preimmagine:** non è possibile trovare un input che, dopo l'applicazione della funzione di hash, fornisca un output specifico. Più precisamente, sono funzioni progettate in modo tale che dato uno specifico input, è computazionalmente impossibile trovare un secondo input che produce lo stesso identico output (es. dato x, trovare y tale che $hash(x) = hash(y)$).

- **Resistenza alla collisione:** non è possibile trovare due input che restituiscono lo stesso output in seguito all'applicazione della funzione di *hashing*. E' quindi computazionalmente impossibile trovare due input che producono lo stesso *digest* in output (es. trova una x ed una y tale che $hash(x) = hash(y)$).

Una specifica Funzione Hash Crittografica usata in molte implementazioni di reti di Blockchain è il *Secure Hash Algorithm (SHA)* con una dimensione di output di 256 bit (da qui, SHA-256).
L'algoritmo sopracitato fornisce un output di 32 bytes (1 byte = 8 bit, 32 bytes = 256 bit), generalmente mostrato come una stringa esadecimale di 64 caratteri.

Ci sono quindi 2^{256} 10^{77}, cioè
115 792 089 237 316 195 423 570 985 008 687 907 853 269 984 665 640 564 039 457 584 007 913 129 639 936 possibili valori di *digest*.

Input Text	**SHA-256 Digest Value**
1	0x6b86b273ff34feel9d6b804eff5a3f5747ada4eaa22fld49c01e52ddb7875b4b
2	0xd4735e3a265e16eee03f59718b9b5d03019c07d8b6c51f90da3a666eec13ab35
Hello, World!	0xdffd6021bb2bd5b0af676290809ec3a53191dd81c7f70a4b28688a362182986f

Essendo il numero di possibili input infinito ed il numero di possibili output finito, è possibile ma molto improbabile ottenere una collisione, cioè $hash(x) = hash(y)$ tale che, dati due input differenti tra loro essi producano lo stesso *digest* in output.

SHA-256 è resistente alle collisioni dato che per trovare una collisione, è necessario eseguire l'algoritmo, in media, 2^{128} volte.

In una rete Blockchain, le Funzioni Hash Crittografiche vengono utilizzate per varie operazioni, tra cui:

- Derivazione dell'Indirizzo

- Creazione di identificatori univoci

- Protezione dei dati del blocco: un nodo pubblicante applicherà le funzioni Hash ai dati del blocco, producendo così un output *digest* unico, che verrà in seguito incluso nell'header del blocco.
- Protezione dell'header del blocco: tramite l'hashing dell'header del blocco e la successiva inclusione del messaggio *digest* ottenuto in output all'interno dell'header del blocco successivo.

2.2.2 Nonce Crittografico

Il *Nonce Crittografico* è un numero arbitrario che è utilizzato una volta sola. Può essere combinato con dati per produrre messaggi di *Hash Digest* differenti tra loro:

$$hash(data + nonce) = digest$$

Un cambiamento del valore di *Nonce*, fornisce un meccanismo per ottenere differenti valori di *digest*, mantenendo però gli stessi dati.

2.2.3 Transazioni

Una *transazione* (2.2) rappresenta una interazione tra due entità. Relativamente alle Criptomonete, una transazione rappresenta un trasferimento della criptomoneta tra due utenti della Blockchain. In uno scenario business-to-business invece, una transazione potrebbe essere un modo di registrare attività avvenute su beni digitali o fisici.

Un blocco in una Blockchain può contenere zero o più transazioni. Per alcune Blockchain, la continua produzione di nuovi blocchi (anche con zero transazioni) è cruciale per mantenere sicura la Blockchain. Infatti, una costante pubblicazione di blocchi previene potenziali utenti malevoli di essere "al passo" e produrre quindi una Blockchain alterata.

Una singola transazione di criptomonete generalmente contiene le seguenti informazioni:

- **Input** - Generalmente, gli input di una transazione corrispondono ad una lista dei *digital asset*[1] da trasferire (es. 1 Bitcoin). Per garantire la provenienza degli asset, la transazione include un riferimento univoco (testuale) relativo alla fonte di provenienza, la quale può essere la transazione precedente, oppure in caso di nuovi asset, l'evento originario. Considerando il

[1] Beni, risorse digitali

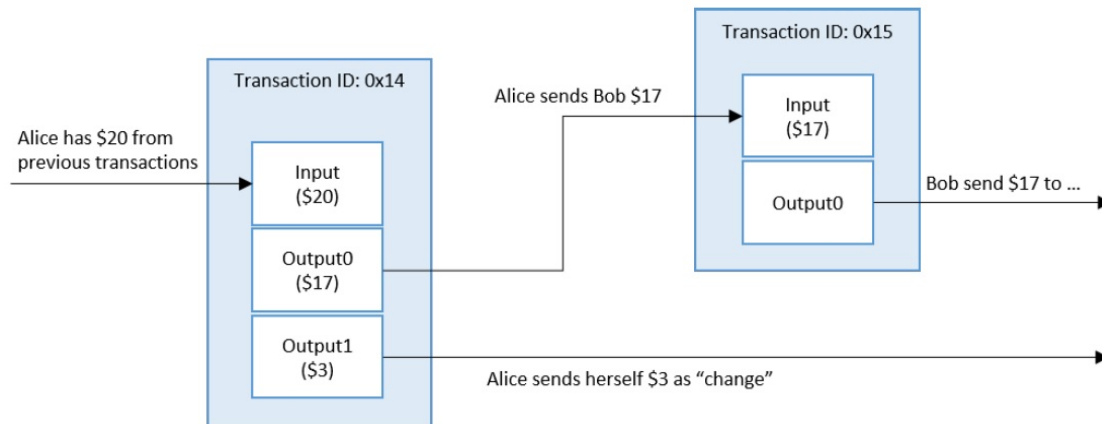

Figura 2.2: Esempio di una Transazione di Criptomoneta [2].

fatto che l'input della transazione è una referenza ad eventi passati, gli asset non possono cambiare.
Questo significa che il valore non può essere aggiunto o rimosso dagli asset esistenti ma può essere solamente suddiviso in molteplici nuovi asset, ognuno con valore minore (es. 1 BTC / 2 = 0,5 BTC e 0,5 BTC). Equivalentemente, molteplici asset possono essere combinati tra loro per formare nuovi asset con valore superiore (0,5 BTC + 0,5 BTC = 1 BTC). Il mittente della transazione deve anche fornire la prova di possedere accesso ai riferimenti forniti in input, generalmente questo avviene firmando digitalmente la transazione. Questo processo fornisce inoltre la prova di avere accesso alla chiave privata del portafoglio.

- **Output** - Gli output della transazione corrispondono all'indirizzo del portafoglio che riceverà gli asset,la quantità di asset che riceverà e un set di condizioni che il nuovo proprietario dovrà rispettare per poterli spendere.

Nonostante le transazioni siano principalmente utilizzate per il trasferimento di *digital assets*, possono anche essere utilizzate più genericamente per trasferire dati.
Ad esempio, qualcuno potrebbe voler pubblicare dei dati sulla Blockchain in modo permanente e pubblico. Nel caso specifico degli Smart Contracts, le transazioni possono essere utilizzare per inviare dati, processarli e poi registrare un eventuale risultato sulla Blockchain.

E' molto importante, in una transazione, verificare la validità ed autenticità di ogni dato scambiato. La validità assicura che la transazione rispetti i requisiti del

protocollo e del formato dei dati. L'autenticità invece, determina e quindi certifica che il mittente degli assets abbia realmente accesso agli stessi.

Le transazioni inoltre, sono firmate digitalmente dalla chiave privata associata al mittente (*Asymmetric-Key Cryptography)* e può essere verificata in ogni momento utilizzando la sua corrispondente chiave pubblica.

2.2.4 Blocchi

Gli utenti della Blockchain inviano transazioni alla rete Blockchain attraverso l'utilizzo di un software, che può essere: un'applicazione desktop, un'applicazione per smartphone, portafogli digitali, servizi web o altro. Il software invia queste transazioni ad uno o più nodi della rete. Il nodo ricevente potrebbe essere di due tipi: nodo non-pubblicante o nodo pubblicante. In seguito, le transazioni vengono poi propagate ad altri nodi della rete e vengono messe in coda fino a che un nodo pubblicante le aggiunge effettivamente alla Blockchain.

Le transazioni in coda vengono aggiunte alla Blockchain solo quando un nodo pubblicante pubblica un blocco. Un *blocco* è composto dal *block header* e dal *block data*. Il *block header* contiene all'interno di se i metadata relativi al blocco. Il *block data* invece, contiene una lista di valide ed autenticate transazioni che sono state inviate alla rete Blockchain.

La validità e l'autenticità è certificata controllando che la transazione sia correttamente formattata e che i mittenti dei *digital assets* l'abbiano firmata digitalmente. Tutti gli altri nodi che ricevono il blocco, verificheranno che tutte le transazioni al suo interno siano valide ed autentiche, in caso di transazioni non valide, il blocco verrà rifiutato.

Ogni Blockchain può avere un'implementazione differente dei propri blocchi e quindi i dati che trasportano, ma in linea generale, molte Blockchain utilizzano il seguente modello:

- Block Header
  - Numero del blocco, conosciuto anche come *altezza del blocco*.
  - Il valore di *hash* del blocco precedente.
  - Il valore di *hash* dei dati del blocco.
  - Un *timestamp*.
  - La dimensione del blocco.

  - Il valore di *nonce*.

- Block Data

  - Una lista delle transazioni o eventi inclusi dentro al blocco.
  - Altri dati potrebbero essere presenti.

2.2.5 Blocchi concatenati

I Blocchi sono concatenati tra di loro tramite la presenza all'interno del *block header* dell' *hash digest* del blocco precedente. In caso un blocco precedentemente pubblicato subisca una modifica (anche a livello di bit), questa modifica cambierebbe totalmente l'*hash digest*. Questo innescherebbe un effetto a catena per tutti i blocchi successivi, in quanto tutti contengono l'*hash digest* del blocco precedente. Questo rende possibile l'identificazione ed il conseguente rifiuto del blocco alterato.

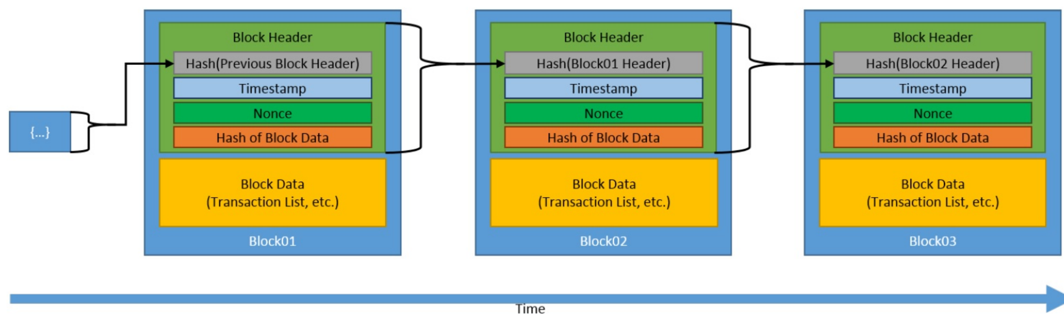

Figura 2.3: Concatenazione dei Blocchi

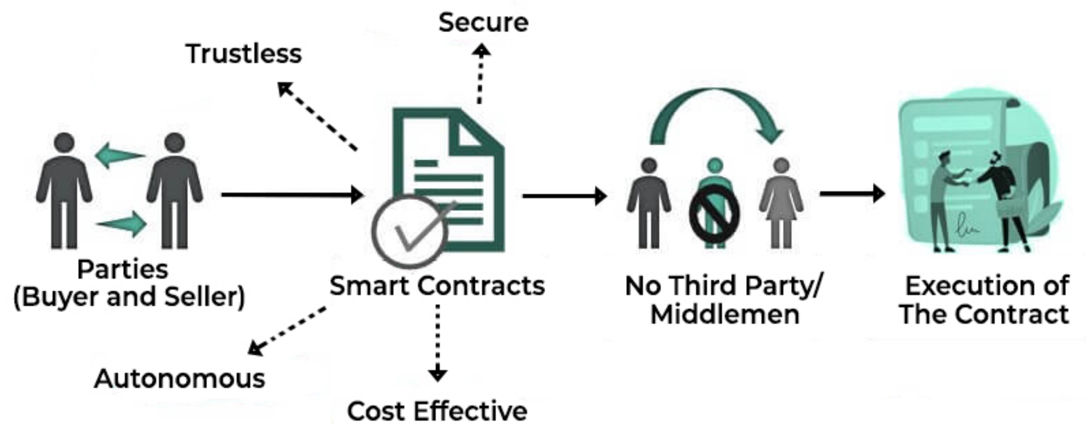

Figura 2.4: Illustrazione di uno Smart Contract

2.2.6 Smart Contracts

Il termine "Smart Contract" risale al 1994, definito da Nick Szabo come "un protocollo di transazione informatizzato che esegue i termini di un contratto. Gli obiettivi generali del design di uno Smart Contract mirano a soddisfare comuni condizioni contrattuali (come metodi di pagamento, privilegi, confidenzialità ed applicazione), minimizzare eccezioni sia malevoli e sia accidentali. Inoltre, minimizzare il bisogno di avere intermediari di fiducia"[6].

Uno Smart Contract (2.4) è una collezione di codice e dati depositati utilizzando transazioni firmate crittograficamente sulla rete Blockchain (es. Ethereum Smart Contract o Hyperlederger Fabric Chaincode). Lo Smart Contract è eseguito dai nodi della rete, di conseguenza, tutti i nodi che eseguono lo Smart Contract devono derivare (ottenere) lo stesso risultato dall'esecuzione. Infine, i risultati dell'esecuzione sono registrati sulla Blockchain.

Gli utenti della Blockchain possono creare transazioni che inviano dati a funzioni pubbliche messe a disposizione dagli Smart Contract. Lo Smart Contract esegue poi il metodo appropriato ai dati forniti dall'utente per svolgere un servizio.

Per natura della Blockchain, il codice che compone gli Smart Contract è *tamper evident* (a prova di manomissione) e *tamper resistant* (antimanomissione, essendo immutabile). Questa caratteristica rende gli Smart Contract affidabili ed è possibile usarli come terze parti di fiducia.
E' importante notare che non tutte le Blockchain supportano e possono eseguire gli Smart Contracts.

Gli Smart Contracts (2.4) sono inoltre deterministici, questo significa che dato un determinato input, produrranno sempre lo stesso output. Inoltre, tutti i nodi che eseguono lo Smart Contract devono approvare il nuovo stato globale che si ottiene dopo l'esecuzione. Per far si che questo accada, gli Smart Contract non possono utilizzare, e quindi operare su dati diversi da quelli che gli si passano in input.

In molte implementazioni di Blockchain, i *Publishing Nodes* (nodi che pubblicano nuovi blocchi) eseguono il codice dello Smart Contract simultaneamente quando pubblicano nuovi blocchi. Sono presenti però alcune Blockchain in cui i Publishing Nodes non eseguono direttamente il codice di uno Smart Contract, ma validano soltanto i risultati dei nodi che eseguono. In Blockchain come quella di Ethereum, gli utenti che effettuano una transazione ad uno Smart Contract, devono pagare per il costo della computazione.

E' presente un limite di quanto tempo computazionale può essere utilizzato da una chiamata allo Smart Contract, ed è basato sulla complessità del codice. Se viene oltrepassato tale limite di tempo, l'esecuzione viene interrotta e la transazione viene scartata.
Questo meccanismo non solo ricompensa chi ha sviluppato lo Smart Contract, ma previene anche utenti malevoli dall'utilizzare gli Smart Contract per eseguire attacchi di tipo Denial of Service consumando tutte le risorse computazionali (es. utilizzando loop infiniti).

Per reti di Blockchain come Hyperledger Fabric Chaincode, invece, potrebbe non esserci il requisito per gli utenti di pagare per l'esecuzione del codice dello Smart Contract. Questo è dovuto al fatto che questo tipo di Blockchain sono costruite per un uso tra partecipanti che si conoscono tra loro ed inoltre, possono essere implementati metodi di prevenzione da "comportamenti scorretti", ad esempio revocando gli accessi.

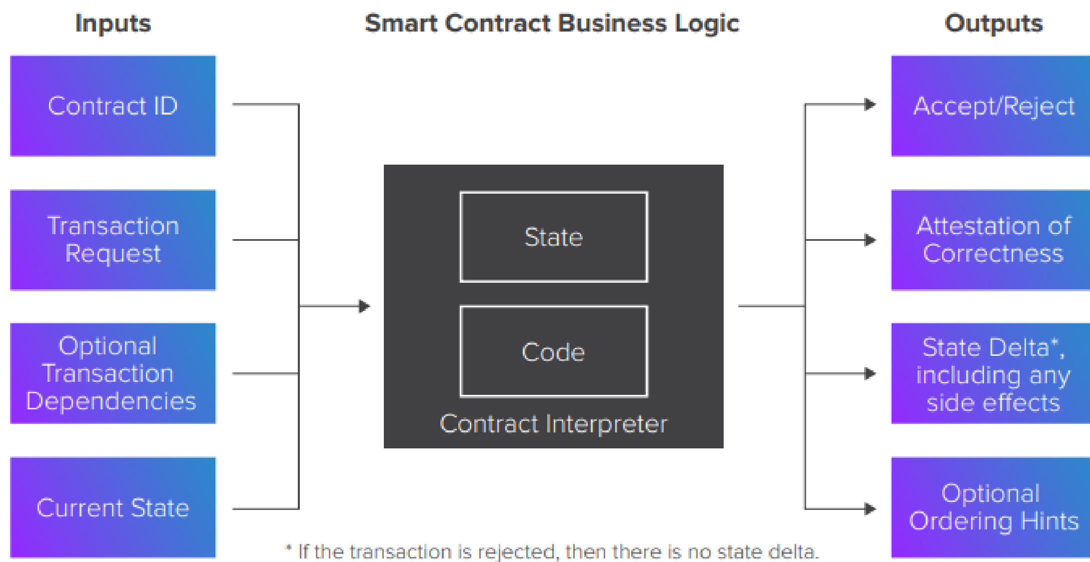

Figura 2.5: Come gli Smart Contract processano le richieste [3].

2.2.7 Processi di uno Smart Contract

Quando viene effettuata una richiesta ad uno Smart Contract, l'input della transazione include un'identificativo del contratto, la richiesta della transazione, ogni dipendenza che potrebbe esistere e lo stato corrente del **ledger** (registro).
Il *Contract Interpreter* (2.5), riceve lo stato corrente del *ledger* ed il codice dello Smart Contract. Quando il Contract Interpreter riceve una richiesta, controlla immediatamente e rifiuta ogni richiesta non valida.

Gli **Output** vengono generati se la richiesta è valida e accettata. Questi output includono un nuovo stato globale ed ogni "effetto collaterale" che si potrebbe verificare. Quando l'elaborazione è completa, il Contract Interpreter impacchetta il nuovo stato, un attestato di correttezza (validità), ed ogni suggerimento utile all'ordinamento per i servizi di consenso. Infine, il pacchetto generato viene inviato al servizio di consenso per la finalizzazione sulla Blockchain.
L'elaborazione di una richiesta può generare "effetti indesiderati", cioè notifiche di eventi, richieste ad altri Smart Contracts, modifiche a risorse globali o anche azioni irreversibili come il rilascio di informazioni sensibili o la spedizione di un pacchetto.

2.3 Consenso

Un aspetto chiave di ogni Blockchain è la scelta dell'*Algoritmo di Consenso*.

L'obiettivo del Consenso in una Blockchain è quello di generare un accordo sull'ordine e convalidare la correttezza dell'insieme delle transazioni che costituiscono un blocco.

Il *Consenso* è il processo per il quale una rete di nodi fornisce una garanzia sull'ordine delle transazioni e la convalida dei blocchi di transazioni.
Deve fornire le seguenti funzionalità chiave:

- Conferma la correttezza di tutte le transazioni proposte in un blocco, secondo politiche di approvazione e consenso.
- Concorda sull'ordine e sulla correttezza e quindi sui risultati dell'esecuzione (implica accordo sullo stato globale).
- Si interfaccia e dipende dallo *Smart Contract Layer* per verificare la correttezza di un set di transazioni ordinate in un blocco.

2.3.1 Tipi di Consenso

Il *Consenso* in una Blockchain può essere implementato in differenti modi: attraverso l'uso di algoritmi *Lottery-Based*[2], tra cui *Proof of Elapsed Time (PoET)* e *Proof of Work (PoW)* o attraverso l'uso di metodi basati sul voto, inclusa la *Redundant Byzantine Fault Tolerance (RBFT)[7]*.

Gli Algoritmi *Lottery-Based* (2.6) sono vantaggiosi in termini di scalabilità, in quanto sono in grado di gestire un elevato numero di nodi. In questo tipo di implementazioni, il "vincitore della lotteria" propone un blocco e lo trasmette agli altri nodi della rete per la convalida.
Durante l'esecuzione di questi algoritmi è possibile che si verifichi una *fork*[3] della Blockchain nel caso in cui due "vincitori" propongano un blocco nello stesso momento. A questo punto, ogni fork deve essere risolto, il che si traduce in un tempo più lungo per ottenere la finalità di un'azione.

Gli Algoritmi *Voting-Based* (2.6) sono vantaggiosi in termini di velocità in quanto forniscono una finalità d'azione a bassa latenza. Quando la maggior parte

[2] Algoritmi basati sul concetto di Lotteria.

[3] In ambito dell'Ingegneria del Software dell'Informatica, un Fork fa riferimento ad uno sviluppo di un nuovo progetto software che parte dal codice sorgente di un altro progetto già esistente.

	Permissioned Lottery-based	Permissioned Voting-based	Standard Proof of Work (Bitcoin)
Speed	GOOD	GOOD	POOR
Scalability	GOOD	MODERATE	GOOD
Finality	MODERATE	GOOD	POOR

Figura 2.6: Tabella Comparativa degli approcci di Consenso Permissioned e Standard Proof of Work [4].

dei nodi valida una transazione o un blocco, il consenso esiste e la finalità occorre. Poiché gli algoritmi *voting-based* richiedono di trasferire messaggi tra i vari nodi della rete, più nodi esistono sulla rete, maggiore sarà il tempo richiesto per raggiungere il consenso.

Il primo passo riguardante il processo di raggiungere il *Consenso* consiste nel ricevere le transazioni dal Client (2.7). Il Consenso dipende fortemente da un *servizio di ordinamento* per ordinare tutte le transazioni che vengono proposte. Il *servizio di ordinamento* può essere implementato in vari modi: sia come servizio centralizzato, che può essere utilizzato durante lo sviluppo ed il testing, sia come protocolli distribuiti che interagiscono con diverse reti e modelli di tolleranza alle rotture.

Per assicurare la confidenzialità delle transazioni, il servizio di ordinamento deve essere agnostico alle transazioni. Questo significa che il contenuto delle transazioni può venire *hashato* o *criptato*.
Le transazioni vengono inviate attraverso un'interfaccia al servizio di ordinamento. Successivamente, il servizio, raccoglie le transazioni basandosi sull'algoritmo di consenso e sulle politiche di configurazione, che potrebbero definire un tempo limite o specificare il numero di transazioni ammesse. Per motivi di efficienza, molteplici transazioni possono essere raggruppate in un unico blocco.

Per convalidare le transazioni, il *Consenso* dipende dal livello dello Smart Contract, poichè esso contiene la logica di business alla base di ciò che rende valida una transazione. Il livello di Smart Contract convalida ogni transazione singolarmente assicurandosi che sia conforme alla politica e al contratto specificato per

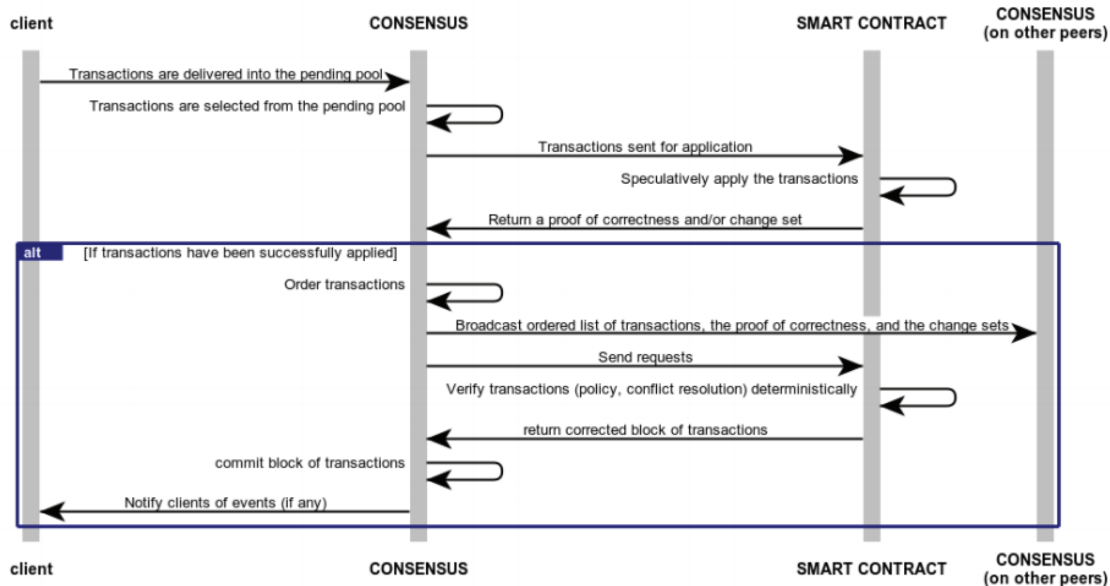

Figura 2.7: Processo di Consenso Generalizzato di Hyperledger [4].

essa. Tutte le transazioni non valide vengono rifiutate ed eventualmente eliminate dall'inclusione all'interno di un blocco.

Possiamo suddividere gli errori di convalida in due categorie: errori di sintassi ed errori di logica. Per errori di sintassi si intendono input non validi, firma non verificabile e transazioni ripetute, le transazioni affette devono essere scartate. Gli errori di logica invece, sono più complessi e dovrebbero essere guidate dalle politiche specificate. Per esempio, una transazione che risulterebbe in *double-spending*[4], deve essere scartata.

2.3.2 Proprietà del Consenso

Il Consenso deve soddisfare due proprietà fondamentali per garantire l'accordo tra nodi: ***safety*** e ***liveness***.

Safety significa che è garantito che ogni nodo riceverà la stessa sequenza di input e darà come risultato lo stesso output in ogni nodo. L'algoritmo di Consenso deve comportarsi in modo identico in ogni nodo che esegue ogni transazione in modo atomico, una alla volta.

[4]Fenomeno per il quale una transazione viene spesa più volte invece che una singola volta.

Liveness significa che ogni nodo non compromesso, quindi completamente funzionante ed operativo, riceverà ogni transazione inviata, supponendo che la comunicazione non fallisca.

Capitolo 3

Hyperledger

Hyperledger è una Community Open-Source che mira allo sviluppo di una suite di framework, strumenti e librerie stabili per implementazioni Blockchain di livello aziendale.

Hyperledger incuba e promuove una vasta varietà di tecnologie Blockchain orientate alle aziende, inclusi i Framework di Ledger (registri) distribuiti. Un Ledger distribuito è un database a più parti senza un'autorità centrale di cui doversi fidare. La differenza principale è che quando le transazioni vengono elaborate in blocchi seguendo l'ordine di una Blockchain, esse danno vita come risultato, ad un Ledger distribuito.

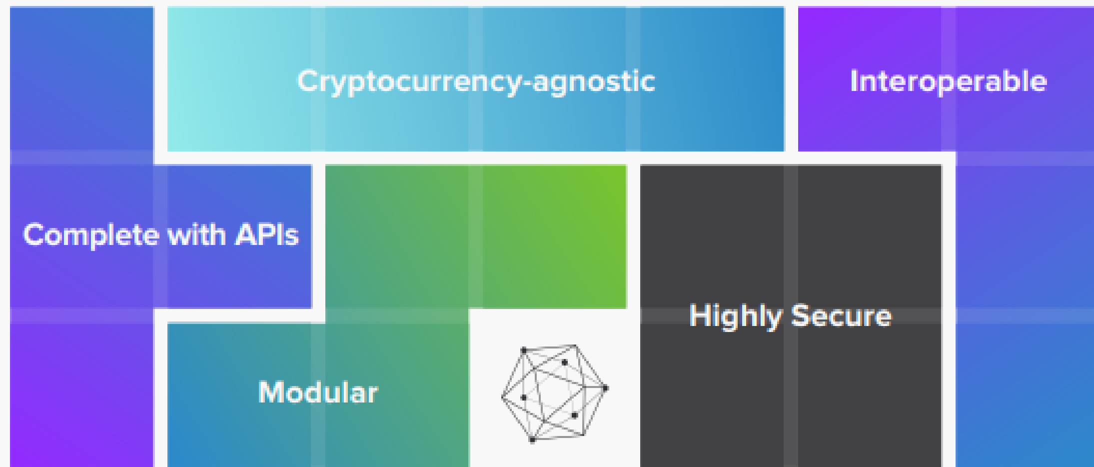

Figura 3.1: Filosofia di Design di Hyperledger [5].

3.1 Architettura di Hyperledger

Tutti i progetti di Hyperledger seguono una “filosofia” per quanto riguarda il design (3.1), il quale include un approccio modulare estensibile, interoperabilità, soluzioni sicure, approccio *Token-Agnostic* senza criptomoneta nativa e lo sviluppo di ricche e facili da usare API (Application Programming Interface).

- **Consensus Layer:** Responsabile per la generazione di un accordo sull'ordine e conferma della correttezza dell'insieme delle operazioni che costituiscono un blocco.

- **Smart Contract Layer:** Responsabile per l'elaborazione delle richieste di transazione e determinare se le transazioni sono valide eseguendo la logica aziendale implementata.

- **Communication Layer:** Responsabile del trasporto dei messaggi *peer-to-peer* tra nodi che partecipano in una istanza di registro condiviso (shared ledger).

- **Data Store Abstraction:** Consente a diversi archivi di dati di essere utilizzati da altri moduli.

- **Crypto Abstraction:** Consente lo scambio di diversi algoritmi o moduli crittografici senza influenzare gli altri moduli.

- **Identity Services:** Consente la creazione di fiducia durante l'installazione di una istanza di Blockchain, della registrazione di identità o entità di sistema durante il funzionamento della rete e la gestione di modifiche come drops, aggiunte e revoche. Inoltre, fornisce l'autenticazione e l'autorizzazione.

- **Policy Services:** Responsabile della gestione delle politiche specificate nel sistema, come la politica di approvazione, la politica di consenso. Si interfaccia e dipende da altri moduli per far rispettare le varie politiche.

- **API:** Consente ai client e alle applicazioni di interfacciarsi con la Blockchain.

- **Interoperation:** Supporta l'interoperabilità tra differenti istanze di Blockchain.

3.2 Progetti Hyperledger

Hyperledger Besu Hyperledger Besu è un Client Ethereum progettato per essere di facile utilizo per i casi d'uso di Reti *Permissioned*[1] sia pubbliche che private. Hyperledger Besu include e supporta diversi Algoritmi di Consenso come il *Proof of Work* (*PoW* ed il *Proof of Authority* (*PoA*).

Hyperledger Burrow Hyperledger Burrow è una distribuzione Blockchain *Single-Binary* progettata per essere semplice, veloce ed adattabile agli sviluppatori. Inoltre, supporta vari tipi di *Smart Contracts*.

Hyperledger è ottimizzata per casi d'uso di implementazioni Blockchain *Permissioned* con Algoritmi di Consenso di tipo *Proof of Stake* (PoS), ma può essere utilizzato anche per reti private o consorzi.

Hyperledger Fabric Hyperledger Fabric è inteso come base per lo sviluppo di applicazioni o soluzioni con Architettura Modulare. Consente ai componenti, ai servizi di consenso e di appartenenza (membership), di essere *Plug-and-Play*, cioè "Collega e Usa".
Può essere utilizzata in diversi casi d'uso e differenti settori o industrie offrendo un approccio al consenso unico che permette di scalare le prestazioni mantenendo la Privacy.

Hyperledger Indy Hyperledger Indy fornisce strumenti, librerie e componenti riutilizzabili per fornire identità digitali sulla Blockchain od altri registri (*ledgers*) distribuiti in modo che siano interoperabili tra diversi domini amministrativi o applicazioni.
Per natura, Indy è *cross-platform*, questo vuol dire che è possibile utilizzarlo con altre Blockchain oppure può essere utilizzato per il la decentralizzazione dell'identità digitale.

Hyperledger Iroha Hyperledger Iroha è una distribuita Blockchain modulare facile da utilizzare. Possiede unici Algoritmi di Consenso e Ordinamento, un ricco modello di autorizzazioni basato sui ruoli e supporto multi-firma (multi-signature).

Hyperledger Sawtooth Hyperledger Sawtooth offre un'architettura flessibile e modulare la quale separa il sistema principale dal dominio dell'applicazione, in modo che gli *Smart Contracts* possano specificare determinate regole per le

[1] Reti controllate da un'Autorità, spesso chi la sviluppa e mantiene.

applicazioni senza dover conoscere il design del sistema principale sottostante. Supporta una varietà di Algoritmi di Consenso, tra cui la *Practical Byzantine Fault Tolerance (PBFT)* e il *Proof of Elapsed Time* (PoET).

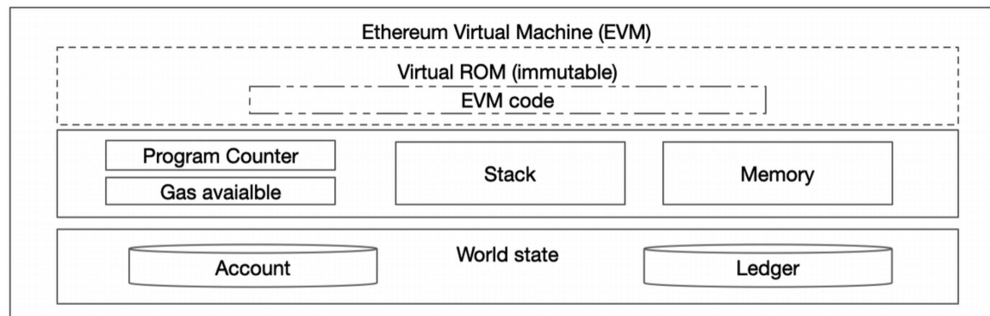

Figura 3.2: Ethereum Virtual Machine

3.3 Hyperledger Besu

Hyperledger Besu è un Client Ethereum Open-Source sviluppato in Java. Può essere eseguito sulla Rete Pubblica di Ethereum, sulle Reti *Permissioned* Private ed anche su Reti Test come Rinkeby, Ropsten e Görli.

3.3.1 Caratteristiche di Hyperledger Besu

Hyperledger Besu implementa la specifica Enterprise Ethereum Alliance (EEA), la quale è stata stabilita per creare interfacce comuni tra i vari progetti *Open e Closed Source* all'interno della Rete Ethereum, per garantire che gli utenti non abbiano alcun vincolo al fornitore e per creare interfacce standard per i team che sviluppano applicazioni.
Le principali caratteristiche sono:

- **Ethereum Virtual Machine (EVM) (3.2):** L'EVM è una completa *Macchina di Turing* virtuale che consente l'implementazione e l'esecuzione di *Smart Contract* tramite transazioni all'interno di una Blockchain di Ethereum.

- **Algoritmi di Consenso:** Implementa vari tipi di Algoritmi di Consenso che sono coinvolti nella convalida delle transazioni, nella convalida dei blocchi e nella produzione dei blocchi (ad esempio, il mining in Proof of Work). Include:

  - **Proof of Authority:** Gli Algoritmi di consenso di tipo Proof of Authority (3.3) sono utilizzati quando i partecipanti di una rete si conoscono tra loro e c'è un livello di fiducia tra essi.

    * **Reti IBFT 2.0:** Le transazioni e i blocchi sono validati e approvati da entità conosciute come *Validatori*. I Validatori prendono

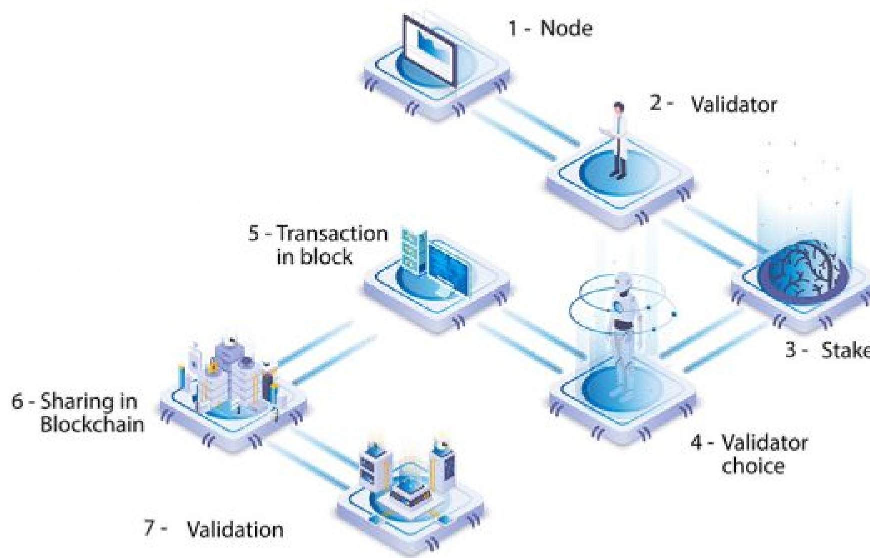

Figura 3.3: Struttura Proof of Authority

decisioni creando il blocco successivo ed i validatori esistenti propongono e di conseguenza, votano, per aggiungere o rimuovere altri validatori.
Le reti IBFT 2.0 hanno finalità immediata, questo vuol dire che non possono avvenire *fork* della rete e tutti i blocchi validi vengono aggiunti alla Blockchain principale.

  * **Clique:** E' più tollerante ai guasti rispetto a IBFT 2.0, infatti tollera fino al fallimento di metà dei validatori mentre le reti IBFT 2.0 richiedono un numero maggiore o uguale ai 2/3 di validatori per essere funzionante e per creare nuovi blocchi. Clique non ha finalità immediata, questo comporta che le sue implementazioni debbano essere consapevoli dei fork e delle riorganizzazioni della Blockchain che si verificano.

- **Proof of Work (Ethash):** E' utilizzato per le attività di mining sulla Blockchain principale di Ethereum (*mainnet*).

- **Storage:** Besu utilizza il database chiave-valore (*key-value*) per rendere persistenti i dati della Blockchain localmente.
I dati possono essere suddivisi in alcune sottocategorie:

  - **Blockchain:** I dati della Blockchain sono composti di *Block Headers* che formano la "catena" di dati utilizzata per verificare crittograficamente lo stato della Blockchain; *Block Bodies* che contengono una lista ordinata della transazioni incluse in ogni blocco; e le ricevute delle transazioni che contengono *metadati* relativi all'esecuzioni delle transazioni inclusi i registri delle transazioni.
  - **World State:** Ogni *Block Header* fa riferimento ad uno stato globale della Blockchain tramite un *Hash stateRoot*. Il *World State* è una mappatura dagli indirizzi ai conti (accounts). Gli Account di proprietà esterna contengono un saldo Ether (ETH), mentre gli *Smart Contract* contengono, inoltre, codice eseguibile e storage.

- **Reti P2P:** Le Reti *Peer-to-Peer* indicano un modello architetturale logico in cui i nodi non sono gerarchizzati unicamente sotto forma di *client* o *server fissi* ma anche sotto forma di *nodi equivalenti o paritari (peer)*, potendo fungere così, sia da *client* e sia da *server* verso gli altri nodi della rete.
  Besu implementa i protocolli di rete di *Ethereum DevP2P* ed anche un addizionale protocollo per *IBFT 2.0*:
  - **Discovery:** Protocollo UDP per la ricerca di *peer* nella rete.
  - **RLPx:** Protocollo TCP per comunicazioni tra *peer* attraverso vari "sotto-protocolli":
    * **ETH Sub-Protocol (Ethereum Wire Protocol):** Utilizzato per sincronizzare lo stato della Blockchain a tutta la rete e per la propagazione delle nuove transazioni.
    * **IBF Sub-Protocol:** Utilizzato dai Protocolli di Consenso di *IBFT 2.0* per facilitare le decisioni.

- **Interfacciamento API:** *Besu* fornisce la *Mainnet Ethereum*[2] e le *API EEA JSON-RPC* via HTTP, via protocolli *WebSocket* e via *API GraphQL*.
  - **JSON-RPC**
    * HTTP JSON-RPC Service
    * WebSocket JSON-RPC Service
  - GraphQL

- **Monitoraggio:** *Besu* permette di monitorare le performance dei nodi e della rete.

[2] Rete Principale di Ethereum

- Le prestazioni dei nodi sono monitorate utilizzando *Prometheus*[3] o il metodo di debug fornito via API, 'debug-metrics JSON-RPC'.
- Le prestazioni della rete sono monitorate con gli strumenti di *Aleth* come *Block Explorer*[4] e *EthStats Network Monitor*[5].

- **Privacy:** In *Hyperledger Besu*, Privacy si riferisce all'abilità di mantenere le transazioni private tra utenti. Altri utenti non possono accedere al contenuto della transazione, al mittente o alla lista di utenti partecipanti. *Besu* utilizza un *Private Transaction Manager* per l'implementazione della Privacy.

- **Permessi:** Una rete *Permissioned* permette solo a specifici nodi e account di partecipare alla rete abilitando l'autorizzazione del nodo e/o l'autorizzazione dell'account sulla rete.

[3] Prometheus è un toolkit Open-Source di monitoraggio e avviso di sistemi

[4] Blockchain Explorer. Strumento utile per recuperare informazioni su blocchi, transazioni, smart contract, ecc.

[5] EthStats è uno strumento utile per monitorare lo stato dell'intera Blockchain di Ethereum.

3.4 Hyperledger Burrow

Burrow è una *Permissioned Smart Contract Machine* che fornisce un *Client Blockchain modulare* con un *Permissioned Smart Contract Interpreter* sviluppato secondo le specifiche della *Ethereum Virtual Machine* (EVM).
Burrow è in grado di eseguire gli *Smart Contract* della *Ethereum Virtual Machine* (EVM) e di *Web Assembly* (WASM).
Le reti di *Hyperledger Burrow* sono sincronizzate utilizzando l'algoritmo di consenso *Tendermint[8]*.

3.4.1 Caratteristiche di Hyperledger Burrow

Burrow possiede un design incentrato all'esecuzione degli *Smart Contract*. Abbina il suo *EVM Execution Engine*[6] al *Tendermint Consensus Engine*[7] su un'interfaccia applicazione-consenso chiamata *ABCI*[8].
Gli account partecipanti alla rete di *Burrow* hanno autorizzazioni e contengono il codice dello *Smart Contract* o corrispondono ad una coppia di chiavi pubblica-privata.

Una transazione che chiama il codice di uno *Smart Contract* di uno specifico account, attiverà l'esecuzione di quel codice all'interno di una *Permissioned Virtual Machine*.

Lo *Smart Contract Engine* di *Burrow* fornisce varie interfacce necessarie per essere usate come librerie.
Alcuni dei suoi sotto-componenti chiave sono:

- **Application Global State:** Lo stato dell'applicazione consiste di tutti gli account, i set validatori e l'integrato registro dei nomi di *Hyperledger Burrow*.
- **Secure Native Functions:** Le funzioni sicure native forniscono le regole di base che tutti gli account e tutti gli Smart Contract devono seguire. Queste regole non risiedono come codice EVM, ma sono esposte alla Permissioned EVM via contratti interfaccia. Hyperledger Burrow supporta l'uso del linguaggio di programmazione nativo per incrementare le performance e la sicurezza.
- **Permission Layer:** La rete è inizializzata con un set di account iniziale con permessi ed un set globale di permessi. I partecipanti della rete con i giusti

[6] Software che esegue codice e/o Smart Contracts.
[7] Algoritmo di consenso Tendermint
[8] Application Blockchain Engine

permessi possono modificare i permessi degli altri account tramite l'invio di transazioni specifiche nella rete. Queste transazioni sono controllate dai validatori di rete prima che le autorizzazioni vengano aggiornate sull'account designato.
Attraverso l'uso dell'EVM, ulteriori, più sofisticate e dettagliate autorizzazioni basate sui ruoli possono essere sfruttate utilizzando i ruoli nativi di *Hyperledger Burrow* su ciascun account. I ruoli degli account possono essere aggiornati mediante transazioni o Smart Contracts. Inoltre, *Burrow* espone la possibilità per gli Smart Contract, all'interno della Permissioned EVM, di modificare il livello di autorizzazione ed i ruoli dell'account. Dopo che uno Smart Contract con questa funzionalità è stato depositato su una *Chain*, un partecipante della rete con gli appropriati permessi, potrà concederre allo Smart Contract tale funzionalità, rendendolo quindi usufruibile.

- **Permissioned EVM:** La *Virtual Machine* (EVM) è costruita per osservare le specifiche del Codice di Ethereum e certificare che siano state concesse le corrette autorizzazioni. Una piccola quantità di *Gas*[9], viene assegnata per ogni transazione. Questo "meccanismo" di commissioni assicura che l'esecuzione di tale operazione verrà completata entro un determinato periodo di tempo. Le transazioni devono essere formulate in formato binario in modo che possano essere elaborate da un nodo della Blockchain attraverso l'utilizzo di un'*Application Binary Interface (ABI).*

- **Gateway:** *Burrow* fornisce ai Client due endpoint, RESTful e JSON-RPC, per l'interazione con la rete Blockchain e lo stato dell'applicazione attraverso la trasmissione delle transazioni suul'intera rete o interrogando lo stato corrente dlel'applicazione.

- **Signing:** Burrow accetta transazioni firmate e formulate dal Client. Inoltre, è disponibile un'interfaccia per la firma remota.

- **Interfaces:** Utilizza anche interfacce di avvio e di runtime, in gran parte tramite file letti dal nodo blockchain all'avvio. *Burrow* include anche una *Remote Procedure Call* (RPC, chiamata di procedura remota), che consente l'interfacciamente con il nodo durante il runtime.

- **Consensus Engine:** Mantiene lo stack di rete tra i nodi e l'ordine delle transazioni che devono essere utilizzate dal *Application Engine* (Motore dell'applicazione).

[9]Piccola commissione per l'esecuzione di ogni operazione su Ethereum

- **Application Blockchain Interface (ABCI):** Fornisce le specifiche dell'interfaccia per la connessione tra il *Consensus Engine* (Motore di Consenso) e l'*Application Engine* (Motore dell'Applicazione).

- **Smart Contract Application Engine:** Fornisce agli sviluppatori uno *Smart Contract Engine* (motore che esegue il codice degli Smart Contract) fortemente deterministico per la gestione di processi complessi.

3.5 Hyperledger Fabric

Fabric è una piattaforma sulla quale è possibile sviluppare *distributed ledger solutions* (registri distribuiti), con architettura modulare la quale fornisce alti livelli di Confindenzialità, Flessibilità, Resilienza e Scalabilità. Questo rende possibile lo sviluppo soluzioni adattabili ad ogni tipo di industria.

Supporta vari Protocolli di Consenso e per questa ragione, può essere adattato a vari casi d'uso e modelli di fiducia.
Esegue applicazioni distribuite scritte in linguaggi di programmazione *general-purpose*[10], senza dipendere da alcuna criptomoneta nativa.

Fabric utilizza una "portabile" nozione di *membership* (appartenenza) per il *Permissioned Model*, che può essere integrato con gli standard d'industria relativi alla gestione delle identità. Supporta inoltre la creazione di "canali", i quali permettono a gruppi di partecipanti di creare un separato registro delle transazioni. Questa funzionalità è importante per le reti dove alcuni partecipanti potrebbero essere dei competitors i quali potrebbero non volere che ogni transazione (come l'offerta di un prezzo speciale a qualcuno, ma non a tutti) sia visibile a tutti i partecipanti della rete. Se un gruppo di partecipanti crea un canale, solo i partecipanti di quel canale avranno una copia del relativo registro delle transazioni.

[10]Linguaggio di Programmazione che può essere utilizzato in differenti scenari, come C, Java, Python, ecc.

3.5.1 Smart Contracts e Caratteristiche di Hyperledger Fabric

Alcuni componenti di *Fabric*, tra cui i servizi di consenso e di appartenenza, sono sviluppati in modo da essere *plug-and-play.* Attraverso l'uso della tecnologia dei *Docker Container*, può ospitare Smart Contract chiamati ***Chaincode***, i quali contengono le regole di business del sistema.

Generalmente, un *Chaincode* gestisce la logica di business, la quale è stata approvata dai membri della rete. Lo stato generato da un Chaincode dopo l'esecuzione di una qualunque operazione, è visibile esclusivamente a quello stesso Chaincode e non può essere visionato direttamente da altri Chaincode nella rete. Tuttavia, attraverso l'uso di determinati permessi, è possibile per un Chaincode richiamarne un altro ed accedere al suo stato.
Ci sono due tipi di *Chaincode* da considerare:

- **System Chaincode:** generalmente gestisce le transazioni relative al sistema come la gestione del ciclo di vita e la configurazione delle politiche. Possiede inoltre API utilizzabili dall'utente per l'implementazione delle applicazioni.

- **Application Chaincode:** gestisce gli stati dell'applicazione sul *ledger*, incluso *digital assets* ed informazioni arbitrarie.

Un *Chaincode* inizia con un pacchetto che incapsula metadati "critici" del Chaincode, tra cui il nome, la versione e le firme digitali che certificano l'integrità del codice e dei metadati.
Un membro della rete attiva il Chaincone inviando un'istanza di transazione alla rete. Se la transazione è approvata, il Chaincode entra nello stato attivo, nel quale può ricevere transazioni dagli utenti attraverso l'uso di applicazioni client-side.

Ogni transazione valida di Chaincode viene successivamente appesa al *ledger*. Queste transazioni possono modificare lo stato globale.

Tipicamente, ci sono due modi per sviluppare un contratto business su *Hyperledger Fabric:*

- Sviluppando individualmente un contratto in una indipendente istanza di Chaincode.

- Utilizzando un Chaincode per la gestione di tutti i contratti (di un certo tipo) ed avere a disposizione delle API per gestire i cicli di vita di quei contratti. Approccio generalmente più efficiente.

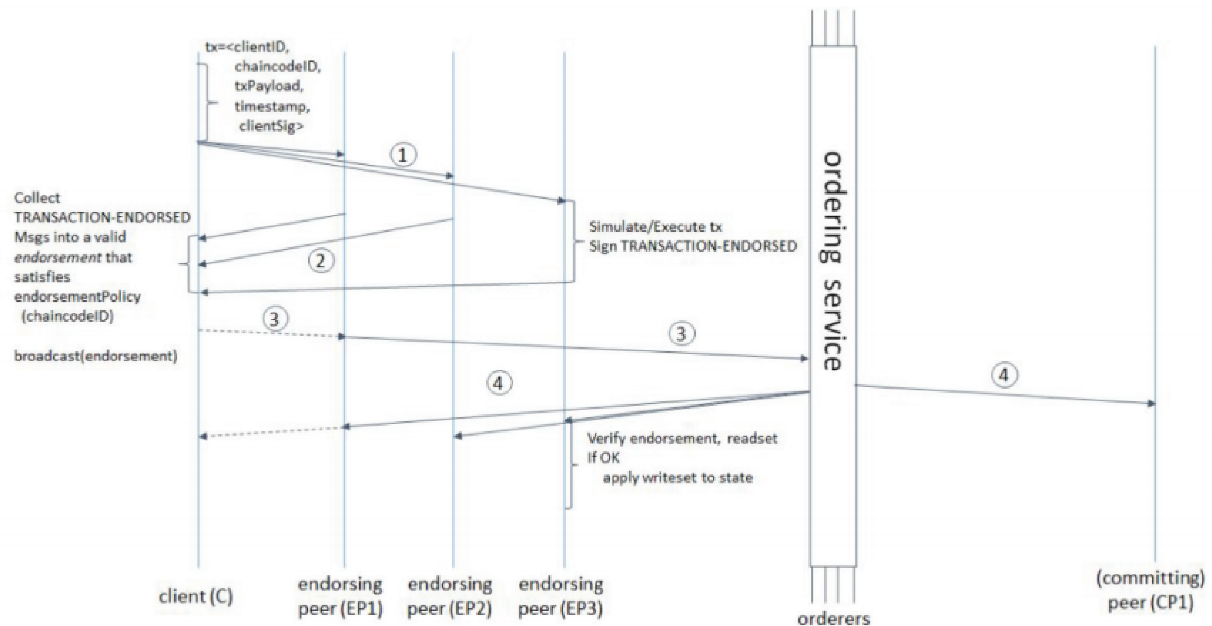

Figura 3.4: Esempio del processo di una possibile transazione in Hyperledger Fabric [4].

3.5.2 Consenso in Hyperledger Fabric

Il Consenso in *Fabric* può essere suddiviso in tre fasi: **Endorsement** (Approvazione), **Ordering** (Ordinamento) e **Validation** (Validazione) (3.4).

- L'*Endorsement* è guidato da politiche (ad esempio m su n firme) in base alle quali i partecipanti della rete approvano una transazione.
- La fase di *Ordering* accetta le transazioni approvate e accetta di inviarle al *ledger*.
- La fase di *Validation* prende un blocco di transazioni ordinate e convalida la correttezza dei risultati, compresa la verifica della politica di approvazione e la doppia-spesa.

Hyperledger Fabric supporta inoltre l'integrazione di servizi di consenso in tutte e tre le fasi. Le applicazioni sviluppate potrebbero avere la necessità di implementare differenti modelli di *endorsement*, *ordering* e *validation*, in base ai loro requisiti.

Le API del servizio di *ordering* consistono in due operazioni basilari: **broadcast** (trasmetti) e **deliver** (consegna).

- **broadcast(blob)**: un client lo richiama per trasmettere messaggi (message blob) ordinari per diffonderli sul proprio canale.
- **deliver(seqno, prevhash, blob)**: il servizio di *ordering* richiama questa funzione sul peer per consegnare il messaggio con la non-negativa sequenza numerica (seqno) e l'hash del messaggio consegnato più recentemente (prevhash).

3.6 Hyperledger Indy

Indy è un *Distributed Ledger* (registro distribuito), progettato per la decentralizzazione dell'identità. Fornisce strumenti, librerie e componenti riutilizzabili per creare ed usare identità digitali indipendenti basate su Blockchain o altri distributed ledgers.
Queste identità sono interoperabili tra domini amministrativi, applicazioni e qualsiasi altro ambito organizzativo. Ciò significa che amici, concorrenti ed anche antagonisti possano fare affidamento su una fonte di verità condivisa.

Il progetto di *Indy* risponde e fornisce soluzioni a domande come "Con chi ho a che fare?" e "Come posso verificare i dati dell'altra persona o entità dall'altra parte di questa interazione?".

3.6.1 Caratteristiche di Hyperledger Indy

Hyperledger Indy non può ospitare gli Smart Contracts. Invece che immagazzinare dati sul *ledger* ed in seguito fornire accesso a quei dati attraverso l'uso di Smart Contracts, Indy consente agli utenti di possedere i propri dati e condividerli in modo da preservare la privacy.

Le identità di Indy possono essere referenziate da Smart Contracts in altri sistemi dando la possibilità di fornire ogni sistema di tipo *distributed ledger* con un sistema d'identità decentralizzato.

Le caratteristiche principali di *Indy* sono:

- **Self-sovereignity:** *Indy* immagazzina gli artifatti delle identità su un *ledger* con proprietà distribuita. Questi artefatti possono includere chiavi pubbliche, prove di esistenza (proof of existence), accumulatori crittografici che consentono la revoca, ecc. Nessuno tranne il vero proprietario degli artefatti può modificare o rimuovere un'identità.

- **Privacy:** per impostazione predefinita, Indy preserva la Privacy poiché ogni proprietario d'identità può operare senza creare alcun rischio di correlazione o breadcrumb[11].

- **Verifiable claims:** gli attestati di identità possono essere ad esempio Certificati di Nascita, Patente di Guida, Passaporti, ecc. L'utilizzo delle prove *zero-knowledge* consente la divulgazione selettiva solo dei dati richiesti ad un particolare contesto.

[11] In informatica, si riferisce al percorso di navigazione o traccia delle azioni eseguite

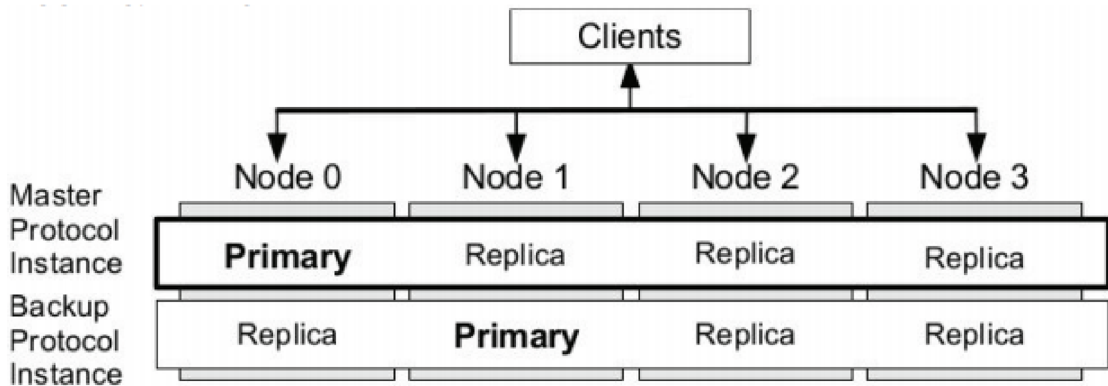

Figura 3.5: Rappresentazione di RBFT [4].

3.6.2 Consenso in Hyperledger Indy

Il Consenso in *Indy* è basato sul protocollo *Redundant Byzantine Fault Tolerance (RBFT)* (3.5), il quale è un protocollo ispirato a *Plenum Byzantine Fault Tolerance (PBFT)*. E' quindi possibile pensare a RBFT come l'esecuzione di multiple istanze di *Plenum* in parallelo. Le richieste Ordinate da una singola istanza chiamata *master* sono utilizzate per aggiornare il *ledger*, però le performance in termini di *throughput* (portata) e latenza sono periodicamente comparate alle performance medie delle altre istanze in esecuzione. Se l'istanza *Master* presenta performance sotto la media, il ruolo di master viene passato ad un'altra istanza.
RBFT richiede almeno 3f + 1 nodi per gestire un nodo difettoso "f".
La figura 3.5 rappresenta una rete composta di 4 nodi che possono tollerare 1 nodo "difettoso" ed ha due istanze in esecuzione, un master e un backup.

La figura 3.6 invece, mostra una vista diversa relativa a RBFT. Il client sta inviando una richiesta ai nodi, la quale dovrà essere inviata ad almeno f+1 nodi. Dopo aver ricevuto la richiesta, i nodi effettuano un processo di propagazione (Propagate) per il quale viene avvisato ogni nodo della rete della richiesta.
Ogni primaria crea una proposta, a partire dalle richieste ricevute, chiamate *PRE-PREPARE* e la invia a tutti gli altri nodi. Se i nodi accettano la richiesta della primaria, inviano un ACK alla proposta tramite un messaggio chiamato *PREPARE*. Non'appena un nodo riceve una proposta *PRE-PREPARE* e 2f messaggi *PREPARE*, allora avrà a disposizione informazioni sufficienti per accettare la proposta ed inviare un messaggio di *COMMIT*. Di conseguenza, quando un nodo riceve 2f+1 messaggi di COMMIT, allora il batch di richieste può venire ordinato ed aggiunto al *ledger* dato che un numero sufficiente di nodi ha concordato che la maggioranza dei nodi ha accettato la proposta.
Non è richiesto il completamento di una proposta prima che si possa inviare

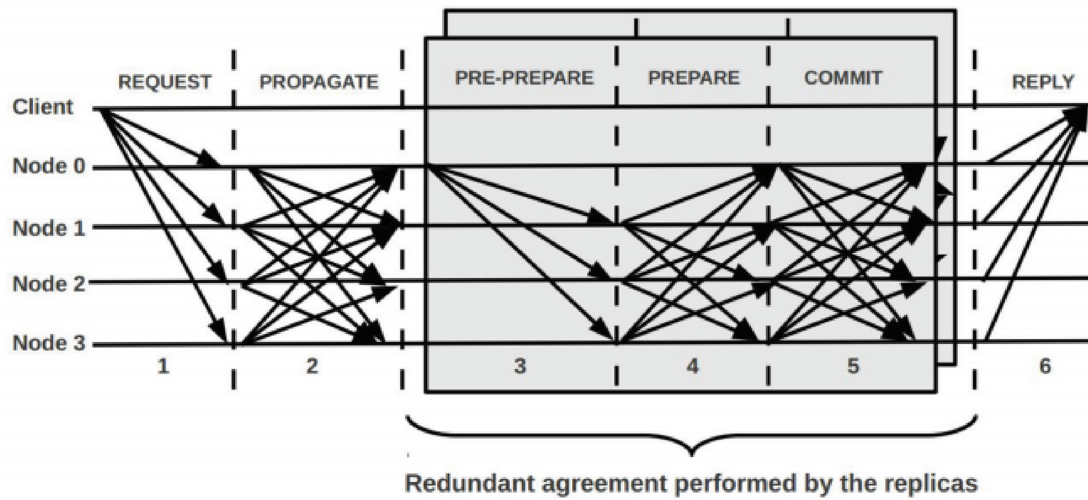

Figura 3.6: Passaggi di un Protocollo RBFT [4].

un'altra proposta.

Indy utilizza RBFT per gestire l'ordinamento e la validazione, portando come risultato un singolo *ledger* contenente tutte le transazioni ordinate e validate.
Plenum, e quindi *RBFT* mantengono proiezioni del *ledger* chiamate stati. Tutte le operazioni valide ed accettate eseguite potrebbero camiare lo stato del *ledger*, il quale è immagazzinato in un database come collezione di variabili ed i loro valori.

3.7 Hyperledger Iroha

Iroha è un Framework Blockchain progettato per essere semplice e facile da integrare in progetti che richiedono l'uso delle tecnologie *distributed ledger*. Essendo un sistema *General Purpose Permissioned Blockchain*, può essere utilizzato per la gestione di *digital assets*, identità e dati serializzati. E' utile quindi per applicazioni come ad esempio, regolamento interbancario, valute digitali della Banca Centrale, sistemi di pagamento, Carte d'Identità o patenti, e logistica.

3.7.1 Caratteristiche chiave di Hyperledger Iroha

- Semplice struttura, sviluppo e manutenzione
- Design Moderno, Domain-Driven C++
- Molteplici librerie per gli sviluppatori
- Gestione di identità ed Assets
- Enfasi, incentrato sullo sviluppo di applicazioni mobile
- Un nuovo, Algoritmo di Consenso Byzantine Fault-Tolerant chiamato ***Sumeragi***

3.7.2 Consenso in Hyperledger Iroha

Iroha introduce un nuovo Algoritmo di Consenso chiamato **Sumeragi**, il quale è un algoritmo di tipo BFT (Byzantine Fault-Tolerant) (3.7).
Consideriamo il concetto di ordine globale sulla convalida dei peer e degli insiemi A e B dei peer, dove A è costituito dai primi 2f+1 peer e B è costituito dal resto dei peer.
Poiché sono necessarie 2f+1 firme per la conferma di una transazione, nel caso normale solo 2f+1 peer sono necessari e coinvolti nella convalida di una data transazione. I peer rimanenti si uniscono alla convalida solo quando si verificano errori nei peer del set A. Il peer numero 2f+1 viene chiamato *Proxy Tail*.

Nei casi normali, quindi senza errori, il flusso della transazione è mostrato come nella figura 3.15 di seguito.

Il client, il quale tipicamente è un API Server che si interfaccia con un client dell'utente finale, all'inizio invia una transazione al *Lead Validating Peer* (peer validatore leader). Il leader a questo punto, verifica la transazione, la ordina nella coda e firma la transazione. Successivamente, la propaga ai rimanenti 2f+1 peer.

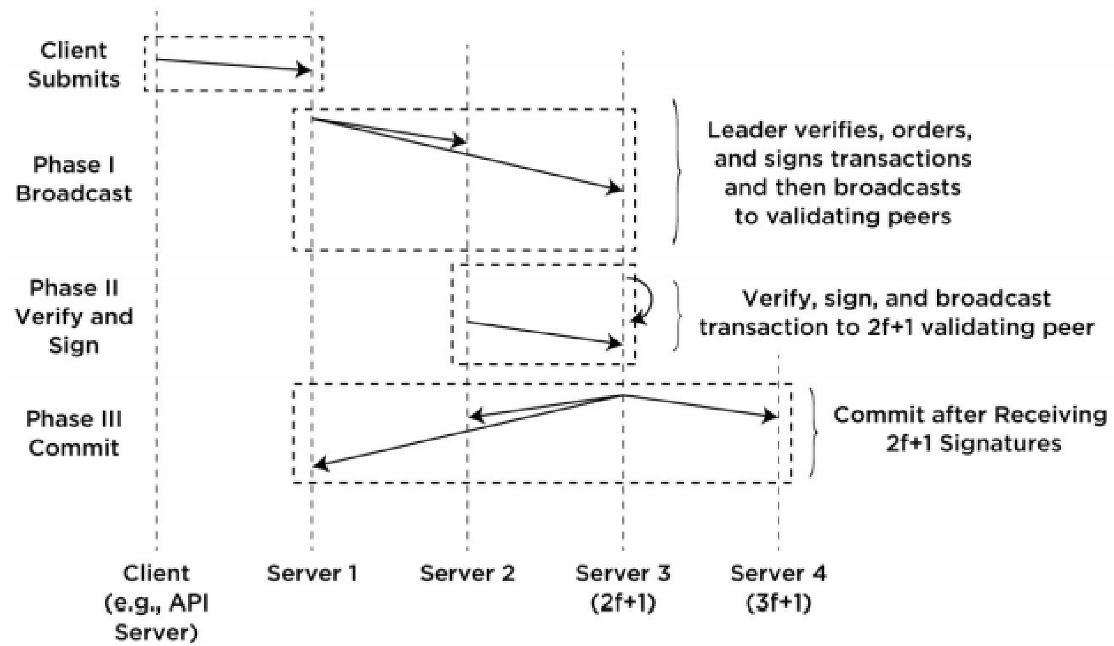

Figura 3.7: Flusso di una transazione in Hyperledger Iroha con Algoritmo Sumeragi [4].

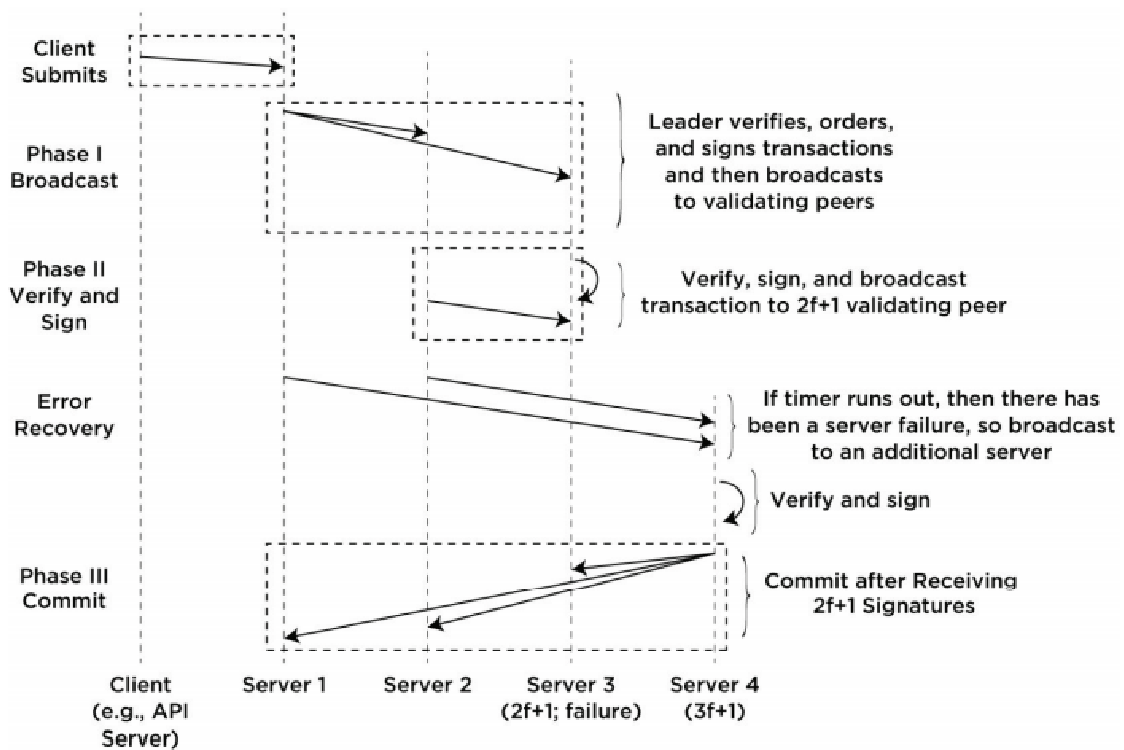

Figura 3.8: Flusso di una transazione in Hyperledger Iroha con Algoritmo Sumeragi nel caso di un Errore nel Server [4].

L'ordine dei nodi elaboratori è determinato in base ad un sistema di reputazione del server chiamato ***Hijiri***, calcolando l'affidabilità dei server in base a tre fattori. Innanzitutto, in base all'ora in cui ogni server si è registrato al servizio di abbonamento. In secondo luogo, in base al numero di transazioni elaborate in passato da ciascun server. Terzo, in base al fatto che siano, o meno, stati rilevati errori.

Per determinare gli errori, ogni server imposta un timer (3.8) quando firma e trasmette una transazione al *proxy tail.* Se si verifica un errore in un server intermedio e non viene ricevuta alcuna risposta prima che il timer scada, allora il server ri-trasmette la transazione e la sua firma al server successivo nella catena dopo il proxy tail.

Il Consenso con l'utilizzo dell'Algoritmo Sumeragi è effettuato su individuali transazioni e sullo stato globale risultante dall'applicazione delle transazione. Quando un peer validatore riceve una transazione sulla rete, esegue in ordine i seguenti passaggi:

- Convalida la firma (o le firme, in caso di transazioni multifirma) della tran-

sazione.

- Convalida il contenuto della transazione, ove presente. Ad esempio, le transazioni di trasferimento devono lasciare il conto di chi paga, con un saldo non negativo.
- Applicare temporaneamente la transazione al *ledger*. Questo comporta l'aggiornamento del *Merkle Root* (radice di Merkle) dello stato globale.
- Firma la *Merkle Root* aggiornata e l'hash del contenuto della transazione.
- Trasmette la tupla, che è un elenco ordinato finito di transazioni.
- Quando si sincronizzano i nodi tra di loro, le parti valide dell'*Albero di Merkle* vengono condivise finchè le radici non corrispondono.

3.8 Hyperledger Sawtooth

Sawtooth è una piattaforma modulare per costruire, depositare ed eseguire *distributed ledgers*. Mira a "mantenere distribuiti i ledger distribuiti" e a far si che gli Smart Contracts siano sicuri da usare per le aziende. Essendo la sua struttura altamente modulare, permette alle aziende, ai business, di prendere decisioni e scegliere come dovrà essere la loro implementazione.

3.8.1 Caratteristiche di Hyperledger Sawtooth

Sawtooth include una serie di innovazioni tecniche tra cui:

- **Dynamic Consensus:** Per *Consenso Dinamico* si intende la possibilità di cambiare l'Algoritmo di Consenso di una Blockchain in esecuzione inviando una semplice transazione.

- **Proof of Elapsed Time (PoET):** E' un Algoritmo di Consenso che possiede la scalabilità dell'algoritmo *Proof of Work (PoW)* ma senza il lato negativo dell'alto consumo di energia elettrica.

- **Transaction families:** E' un'astrazione di uno Smart Contract che permette agli utenti di scrivere la logica degli Smart Contracts nel loro linguaggio preferito.

- **Compatability with Ethereum Contracts:** Le *transaction families* possono integrare altri *Smart Contract Interpretes* come, ad esempio, *Hyperledger Burrow EVM*[12]. Caratteristiche come il *permissioning* e la possibilità di "scollegare" (un-plug) l'algoritmo di Consenso permettono di configurare la Blockchain di Ethereum in base alle esigenze.

- **Parallel transaction execution:** La maggior parte delle Blockchain hanno bisogno di eseguire le transazioni in serie per garantire il costante ordinamento ad ogni peer. *Sawtooth* invece, include un avanzato *Scheduler Parallelo* che divide i blocchi in flussi paralleli. Il parallelismo permette quindi di ottenere una velocità di elaborazione dei blocchi più elevata per compensare le differenze di performance delle Blockchain rispetto ai Database.

- **Private transactions:** Insiemi di nodi possono essere facilmente implementati con permessi separati dagli altri nodi già esistenti nella Blockchain, fornendo Privacy e Confidenzialità tra partecipanti della rete.

Sawtooth supporta due tipi di Smart Contracts: *installed* ed *on-chain*.

[12]Ethereum Virtual Machine di Hyperledger Burrow

3.8.2 Installed Smart Contracts con Transaction Families

Ogni linquaggio programmabile presenza dei rischi e per limitarli, alcune Blockchain specificano una semantica fissa per le transazioni, Queste reti utilizzano le *families of transactions (famiglie di transazioni)* e supportano solo determinate operazioni consentite.

Un esempio è l'*IntegerKey Transaction Family*, la quale fornisce solo tre operazioni: Increment, Decrement e Set. Con solo tre operazioni e nessun loop, questa family aiuta a prevenire ogni intenzionale o accidentale problema di script.

Un altro esempio è la *Settings Transaction Family*, la quale può essere usata per controllare la rete Blockchain, incluse le specifiche come l'algoritmo di Consenso utilizzato.

Ogni *transaction family* può essere implementato con *Sawtooth*, è solo necessario che questa family supporti le API delle *transaction family*. Queste API supportano solo poche operazioni come **get state** per ottenere qualcosa dal *ledger* e **set state** per impostare qualcosa nel *ledger*.

3.8.3 On-Chain Smart Contracts con Seth Transaction Family

Gli *On-Chain Smart Contracts* sono gestiti grazie all'integrazione della *Hyperledger Burrow Ethereum Virtual Machine (EVM)* su un nodo validatore di *Hyperledger Sawtooth*. Dopo l'integrazione, gli Smart Contracts possono essere scritti in *Solidity* utilizzando la *Sawtooth-Ethereum (Seth) Transaction Family* ed il *transaction processor (elaboratore delle transazioni)*. Questa integrazione rende gli Smart Contracts completamente programmabili.

Il comando **seth** può essere utilizzato per interagire con la *Seth transaction family* attraverso l'utilizzo dell'Interfaccia a riga di comando (CMD). Permette il caricamento e l'esecuzione degli Smart Contracts, l'invio di query per i dati associati al contratto e la generazione di chiavi nel formato utilizzato da Seth.

In *Sawtooth* i contratti sono compilati utilizzando il comando **seth load**, il quale ha un attributo **-init** che prende come argomento un array di byte esadecimali. Questa stringa è interpretata come codice di creazione del contratto. Per un *Solidity Smart Contract* è possibile utilizzare il compilatore **solc** per generare questa stringa.

I contratti in Solidity, che possono essere compilati in *Hyperledger Sawtooth*, sono simili alle classi dei linguaggi di programmazione ad oggetti. Possiedono infatti i dati persistenti in variabili di stato e funzioni che possono modificare queste variabili.

3.8.4 Consenso in Hyperledger Sawtooth

Sawtooth facilità l'integrazione di algoritmi di consenso di tipo *lottery-based* e *voting-based*. Come impostazione predefinita, *Sawtooth* utilizza un algoritmo di Consenso *lottery-based*, il **PoET** (Proof of Elapsed Time), il quale utilizza un sistema di elezione basata sul concetto di *guaranteed wait time* (tempo di attesa garantito) attraverso il *Trusted Execution Environment (TEE)* per eleggere un *election leader* (nodo vincitore delle elezioni).
Per ottenere un consenso distribuito in modo efficiente, un algoritmo di tipo *lottery-based* deve avere alcune caratteristiche:

- **Fairness:** La funzione deve distribuire al maggior numero di partecipanti, il risultato dell'elezione, quindi l'*election leader*.

- **Investment:** Il costo per controllare il processo dell'elezione del leader dev'essere proporzionale al valore guadagnato dal fare l'elezione.

- **Verification:** Deve essere relativamente facile per tutti i partecipanti di verificare che il leader sia stato selezionato legittimamente.

La corrente implementazione di *Hyperledger Sawtooth* è costruita su un *Trusted Execution Environment (TEE)* fornito da Intel Software Guard Extensions (SGX). Questo garantisce la sicurezza e la casualità nel processo di elezione del leader senza la necessità di utilizzare ingenti quantità di elettricità.

Ogni *PoET validator* richiede un tempo random di attesa da una funzione affidabile prima di richiedere la leadership. Il validator con minor tempo d'attesa per un particolare blocco è implicitamente eletto come leader. La funzione "CreateTimer" crea un timer per un blocco di transazioni ed è garantito che il timer è stato creato dalla *TEE*. La funzione "CheckTimer" verifica che il timer sia stato creato proprio dalla *TEE* e, se è scaduto, crea un attestato che può essere utilizzato per verificare che il validatore abbia realmente atteso per il tempo specificato prima di richiedere la leadership.

L'utilizzo dell'algoritmo PoET distribuisce casualmente l'elezione tra tutti i validatori con una distribuzione simile a quella fornita da altri algoritmi *lottery-based*. La probabilità di elezione è proporzionale alle risorse con le quali si contribuisce

alla rete. Un'attestazione di esecuzione fornisce informazioni per la verifica che il certificato sia stato creato all'interno di una TEE e che il validatore abbia atteso il tempo assegnatoli. Inoltre, il basso costo per partecipare incrementa la possibilità che il numero di validatori cresca nel tempo, rendendo più robusto l'algoritmo di Consenso.

Capitolo 4

Conclusioni

L'obiettivo centrale di questa tesi non era soltanto indagare lo stato dell'arte della tecnologia alla base della Blockchain ma anche garantire una dettagliata panoramica dei vari principali progetti di Hyperledger. Nel secondo capitolo (2), sono stati descritti i componenti principali di una Blockchain ed il suo funzionamento a partire dalle Funzioni Hash Crittogratiche, fino ad arrivare agli algoritmi di Consenso. La Blockchain infatti, è una tecnologia che esiste da alcuni anni, ma solo recentemente ha iniziato a suscitare interesse sia nelle aziende e sia negli utenti finali, grazie alle sue uniche caratteristiche. Essendo la Blockchain Open-Source, le potenzialità offerte ad oggi sono molteplici ed in continua crescita ed espansione grazie al continuo sviluppo da parte di sviluppatori indipendenti, i quali possono contribuire scrivendo codice, aggiungendo funzionalità o proponendo modifiche o migliorie. Nel terzo capitolo (3) invece, è stato descritto il progetto e l'architettura di Hyperledger per poi passare ad una descrizione dettagliata dei suoi frameworks principali, illustrando le principali caratteristiche di ognuno, il funzionamento degli Smart Contracts e gli algoritmi di Consenso. I Frameworks sviluppati dalla community di Hyperledger sono adattabili a moltissimi scenari nell'ambito aziendale e sono volti a semplificare processi che richiedono fiducia tra le parti che interagiscono.

Il numero di possibili applicazioni ed implementazioni che è possibile sviluppare con la tecnologia della Blockchain, sono innumerevoli. Quasi ogni settore ed industria presente al momento della scrittura di questa tesi, potrebbe essere rivoluzionata attraverso l'implementazione della Blockchain. Prendendo in considerazione il settore immobiliare, è possibile registrare il Contratto d'acquisto di un immobile sulla Blockchain attraverso l'uso di uno Smart Contract, non avendo così la necessità della registrazione di un contratto da parte di una figura autorevole, in questo caso, il Notaio.

Bibliografia

[1] Michael Nofer, Peter Gomber, Oliver Hinz, and Dirk Schiereck. Blockchain. *Business & Information Systems Engineering*, 59(3):183–187, 2017.

[2] Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. Blockchain technology overview. Oct 2018.

[3] Hyperledger. Hyperledger Architecture WG Paper 2 Smart Contracts. https://www.hyperledger.org/wp-content/uploads/2018/04/Hyperledger_Arch_WG_Paper_2_SmartContracts.pdf, 2021. [Online; accessed 03-July-2021].

[4] Hyperledger. Hyperledger Architecture WG Paper 1 Consensus. https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf, 2021. [Online; accessed 03-July-2021].

[5] Hyperledger. Hyperledger Whitepaper Introduction. https://www.hyperledger.org/wp-content/uploads/2018/08/HL_Whitepaper_IntroductiontoHyperledger.pdf, 2021. [Online; accessed 03-July-2021].

[6] Nick SZabo. Smart Contracts. https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html, 1994. [Online; accessed 03-July-2021].

[7] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. Rbft: Redundant byzantine fault tolerance. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*, pages 297–306, 2013.

[8] Tendermint. Tendermint Documentation. https://docs.tendermint.com/master/introduction/what-is-tendermint.html, 2021. [Online; accessed 03-July-2021].

[9] Hyperledger. Hyperledger Besu. `https://www.hyperledger.org/use/besu`, 2021. [Online; accessed 03-July-2021].

[10] Hyperledger. Hyperledger Burrow. `https://www.hyperledger.org/use/hyperledger-burrow`, 2021. [Online; accessed 03-July-2021].

[11] Hyperledger. Hyperledger Fabric. `https://www.hyperledger.org/use/fabric`, 2021. [Online; accessed 03-July-2021].

[12] Hyperledger. Hyperledger Indy. `https://www.hyperledger.org/use/hyperledger-indy`, 2021. [Online; accessed 03-July-2021].

[13] Hyperledger. Hyperledger Iroha. `https://www.hyperledger.org/use/iroha`, 2021. [Online; accessed 03-July-2021].

[14] Hyperledger. Hyperledger Sawtooth. `https://www.hyperledger.org/use/sawtooth`, 2021. [Online; accessed 03-July-2021].