Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Blockchain of Things

Soluzioni di problemi complessi applicando algoritmi di AI su architetture Blockchain

Emilio Greco, Sabrina Celia, Antonio Francesco Gentile

RT- ICAR-CS-21-07 **Settembre 2021**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 8-9C, 87036 Rende, Italy, URL: www.icar.cnr.it
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: www.icar.cnr.it
– Sezione di Palermo, Via Ugo La Malfa, 153, 90146 Palermo, URL: www.icar.cnr.it

Sommario

Premessa

La prima blockchain fu introdotta nel 2008 ad opera di Satoshi Nakamoto. Ha raggiunto una certa notorietà a livello globale a partire dal 2014 dove la dimensione della sua blockchain "**Bitcoin**" raggiunse i 20 gigabyte. Solo nell'aprile del 2019 è stato presentato il primo manufatto artigianale made in Italy, nel quale sono stati tracciati interamente i passaggi produttivi tramite tecnologia blockchain. L'implicazione della **blockchain nell'Industria 4.0** ha generato una grande quantità di innovazioni che hanno consentito la realizzazione di nuovi modelli di business ottimizzati, flessibili e più efficienti, basati sulla fiducia e la sicurezza di tutte le parti interessate. In tale contesto la tecnologia è di grande aiuto ove i consumatori finali sono sempre più interessati a scoprire l'esatta tracciabilità dei prodotti acquistati ma anche per le istituzioni sempre più severe nei controlli di filiera e dei processi di produzione. Oggi non può che suscitare particolare interesse da parte della comunità scientifica che la annovera tra le tecnologie emergenti più promettenti. Con la blockchain viene per la prima volta definito un ecosistema del tutto decentralizzato, privo di autorità centrale che controlli gli scambi informativi e che possa quindi teoricamente modificarne il contenuto o nascondere parte di esse. L'aspetto straordinario di Bitcoin è, che non è di nessuno: nessuna società, nessuno stato la possiede è semplicemente un programma "**Open Source**" operante su Internet. Contemporaneamente allo sviluppo della tecnologia blockchain, l'IoT **(Internet of Things)** ha trovato campo di applicazione in una quantità enorme di applicazioni industriali, al fine di effettuare il monitoraggio e il controllo da remoto o automatico di sistemi elettronici. La peculiarità dell'IoT sta nell'intuizione di collegare ad internet direttamente i dispositivi che raccolgono dati o che effettuano operazioni di controllo. La commistione delle due tecnologie sopracitate ha preso il nome di **Blockchain of Things (BCoT)**, e rappresenta l'ultima evoluzione dello scambio informativo tra dispositivi IoT, che sono quindi in grado, oltre che di effettuare operazioni sull'ambiente, anche di certificare i dati raccolti in maniera automatica tramite la blockchain e fornirli all'utente finale all'interno di un ledger decentralizzato. Oltre a fornire servizi evoluti di "comunicazione", oggi si indaga sulla possibilità di integrazione tra blockchain ed altri sistemi distribuiti, come ad esempio i sistemi robotici a sciame, per fornire a quest'ultimi le capacità necessarie per realizzare sistemi di controllo decentralizzati di **swarm robotics**, più sicuri, autonomi e flessibili. In tale ambito i dispositivi non comunicano con l'essere umano per richiedere comandi o istruzioni ma sono in grado di comunicare tra loro in maniera autonoma ed effettuare operazioni, anche complesse, prendendo decisioni in autonomia, a volte supportati da algoritmi di Machine Learning o Intelligenza Artificiale. In questo lavoro viene fatta una disamina delle tecnologie attualmente presenti, analizzandone le peculiarità al fine di individuare sistemi e soluzioni a problemi complessi applicando algoritmi di AI come quelli di Swarm Intelligence su una rete BCoT (Blockchain of Things).

Introduzione

La tecnologia blockchain può fornire soluzioni innovative a quattro problemi noti nel campo di ricerca della swarm robotics come la gestione della sicurezza, la realizzazione di modelli decisionali, la differenziazione del comportamento e lo sviluppo di modelli di business validi.

Uno dei principali ostacoli allo sviluppo di applicazioni commerciali su larga scala per la swarm robotics è **la sicurezza**. La ricerca nel settore ha evidenziato come vi sia la necessità di sviluppare sistemi in cui i membri di uno sciame debbano potersi fidare delle loro controparti per poter raggiungere l'obiettivo. Questo è particolarmente importante, dal momento che è stato dimostrato che l'inclusione di membri "difettosi" nello sciame o elementi che hanno volutamente intenzioni malevoli potrebbero essere un potenziale rischio anche per gli obiettivi che deve raggiungere l'intero sciame. La sicurezza deve essere quindi garantita in qualsiasi ambiente, sciame compreso, e riguarda fondamentalmente la fornitura di servizi che rispettino: la riservatezza, l'integrità e l'origine dei dati, nonché l'autenticazione dell'entità che li genera. A differenza di altri campi in cui la ricerca in materia di sicurezza viene condotta attivamente, i sistemi robotici a sciame soffrono della mancanza di soluzioni a causa delle caratteristiche complesse ed eterogenee dei sistemi come: l'autonomia del robot, il controllo decentralizzato, un numero elevato degli attori, il comportamento collettivo emergente, ecc. La tecnologia blockchain può quindi fornire non solo un canale di comunicazione peer-to-peer affidabile tra gli agenti dello swarm, ma è anche un modo per superare potenziali minacce, vulnerabilità e attacchi.

Gli **algoritmi decisionali distribuiti** hanno svolto un ruolo cruciale nello sviluppo di sistemi di swarm robotics. Uno degli esempi più importanti è stata la realizzazione della rete ad hoc MANET sviluppata per testare applicazioni di rilevamento distribuito. Questi sistemi hanno la capacità di rilevare le stesse informazioni da più punti e, quindi, di aumentare la qualità dei dati ottenuti. Tuttavia, i robot nello sciame hanno bisogno di raggiungere un accordo globale sull'oggetto di interesse, ad esempio, sui percorsi da attraversare, sulla forma di un oggetto da rilevare o sugli ostacoli da evitare. Pertanto, è necessario sviluppare protocolli decisionali distribuiti che garantiscono la convergenza verso un risultato comune. Gli algoritmi decisionali distribuiti sono stati adottati in molte applicazioni robotiche, tra cui l'allocazione dinamica delle attività, la costruzione di mappe collettive e l'elusione degli ostacoli. Tuttavia la dislocazione di grandi quantità di agenti che fanno uso di un processo decisionale distribuito rappresenta ancora un problema aperto in quando per risolverlo, allo stato attuale è necessario adottare il noto compromesso nel bilanciare velocità di elaborazione ed accuratezza. In questo contesto la blockchain è una tecnologia eccezionale per garantire che tutti i partecipanti di una rete decentralizzata possano condividere una visione identica del mondo. Ogni

volta che un membro dello sciame si trova in una situazione che richiede un accordo, può emettere una transazione speciale, creando un indirizzo associato a ciascuno delle possibili opzioni che lo sciame robotico deve votare. Dopo essere state incluse in un blocco, le informazioni sono disponibili pubblicamente, quindi gli altri membri dello sciame, possono votare in base alla loro situazione, ad esempio, trasferendo un token a l'indirizzo corrispondente all'opzione scelta. Il raggiungimento di un accordo come ad esempio attraverso la regola della maggioranza, può essere ottenuta rapidamente e in un modo sicuro e verificabile poiché tutti i robot possono monitorare gli indirizzi coinvolti nel processo di voto.

Anche se gli algoritmi allo stato dell'arte hanno consentito a team di robot specializzati di **gestire comportamenti collettivi specifici** come aggregazione, raggruppamento, foraggiamento, ecc. ancora non siamo in grado di gestire applicazioni nel mondo reale. Si possono verificare alcuni scenari dove lo sciame deve gestire comportamenti diversi in funzione dell'ambiente, ad esempio, commutando da un algoritmo di controllo all'altro per raggiungere un determinato obiettivo. La combinazione di diversi comportamenti in uno sciame è ancora oggetto di studio in letteratura. In questo caso, la tecnologia blockchain offre la possibilità di collegare diverse blockchain in modo gerarchico, note anche come catene laterali ancorate, che consentirebbero agli agenti di agire in modo diverso a seconda della particolare blockchain in uso, con diversi parametri, con diversità dei miners, permessi, ecc., personalizzando per l'appunto i diversi comportamenti dello sciame.

Il termine blockchain viene associato generalmente ad applicazione per gestire una valuta, o per essere più precisi una "cripto valuta", ma è grazie al lavoro di Vitalik Buterin, fondatore di Ethereum, che nel 2014 si dà il via ad una blockchain di seconda generazione introducendo gli **smart contracts.** Ethereum ha dato il via a numerosi progetti, oggi si parla della quinta generazione della tecnologia. La più recente implementazione della Blockchain riguarda il così detto **Web 3.0** noto anche come **cryptointernet**. Il Web 3.0 nasce come contraltare allo strapotere GAFA (Google, Apple, Facebook, Amazon) i cui modelli di business si basano sulla centralizzazione dei dati e del conseguente potere decisionale. In questa eccezione, la tecnologia blockchain, può essere vista come un'**Interfaccia di programmazione per una applicazione (API)**, ideale sicuramente per applicazioni economiche, ma anche come framework per consentire a sciami di robot di accedere direttamente e partecipare ad un'economia. Per questo motivo la tecnologia blockchain ha il potenziale per stimolare l'uso della robotica a sciame in applicazioni nell'ambito industriale e di mercato. Una delle implementazioni prototipiche più ovvie per quanto riguarda l'uso di sciami robotici in applicazioni economiche è il processo di scambio dati in cambio di valuta tra un robot ed un richiedente. Questo nuovo modello di business emergente in campo dell'Internet delle cose (IoT) prende il nome di **Sensing-as-a-Service**.

SaaS aiuta a creare mercati multiformi per i dati prodotti dai sensori in cui uno o più clienti, il lato acquirente dei mercati, si sottoscrivere per pagare i dati forniti da uno o più sensori, lato vendita.

Anche se la combinazione della tecnologia blockchain e la robotica a sciame può fornire soluzioni utili per affrontare in maniera efficace diversi problemi, occorre ancora lavorare per risolvere diverse sfide tecniche legate alla blockchain al fine di aumentarne l'efficienza.

La latenza è il principale problema su cui si sta lavorando, attualmente con la versione più utilizzata di blockchain, Bitcoin, un blocco impiega circa 10 minuti per essere elaborato. Ciò significa anche che ogni singola transazione richiede circa 10 minuti per essere confermata. Anche se questo problema può essere notevolmente ridotto attraverso l'uso di blockchain private e/o tramite l'utilizzo di diverse politiche di mining, come il proof-of-stake, questo consentirebbe di ottenere prestazioni accettabili **solo per applicazioni peer-to-peer che richiedono una bassa iterazione** con gli utenti finali. Il problema della latenza diventa notevolmente rilevante quando interessa invece applicazioni di swarm robotics, in questo caso, sono necessarie informazioni rapide e affidabili per orchestrare i movimenti dello sciame. Potrebbero sorgere collisioni o altri inconvenienti in situazioni in cui c'è una discrepanza tra lo stato attuale dell'ambiente e quello invece che è stato rilevato o attuato da una transazione. Una possibile soluzione per mitigare questo problema potrebbe essere la creazione basata sull'affiliazione di sistemi robotici appartenenti alla stessa organizzazione in modo che gli elementi appartenenti ad un gruppo non sono tenuti ad aspettare lunghi periodi di tempo per accettare o elaborare transazioni che riguardano l'intero sistema. Potrebbe essere costruito un sistema di reputazione basato su elenchi di precedenti transazioni accettate all'interno del gruppo per ridurre questi tempi di attesa. Altre soluzioni fanno uso della crittografia per stabilire collegamenti off-chain tra i due nodi peer, utilizzando la chiave pubblica del richiedente ed incapsulamento nel campo dati di una transazione. La comunicazione off-chain in questo caso previene la congestione della blockchain e garantisce che solo il richiedente può leggere il messaggio previsto. La blockchain verrà usata solo per finalizzare un accordo e non per lo scambio di dati tra i due contraenti.

Un secondo problema da affrontare nell'uso della tecnologia per la swarm robotics è legato alle **dimensioni, throughput e larghezza di banda**. Se grandi quantità di robot dovessero essere impiegate per un lungo periodi di tempo, potrebbero espandere la blockchain al punto tale da diventare troppo grande per poterne conservare una copia sui singoli nodi. Questo problema, che la comunità Bitcoin chiama "**bloat**", è di particolare rilevanza nella robotica a sciame dove i singoli robot hanno capacità hardware limitate.

In questo primo lavoro cercheremo di indagare se la tecnologia disponibile allo stato dell'arte riesce a rispondere alle sfide che sono state descritte, mettendo in luce i lavori che si stanno conducendo in tale senso.

Nel primo capitolo descriveremo in modo più dettagliato una blockchain, le piattaforme attualmente più utilizzare e le tecnologie abilitanti. Nella seconda parte analizzeremo alcuni algoritmi di swarm intelligence e le loro implicazioni sulle infrastrutture di calcolo, infine, verranno descritte alcune piattaforme utilizzate per affrontare degli studi di fattibilità.

Blockchain

La Blockchain costituisce uno degli sviluppi tecnologici in ambito Internet of Things (IoT). L'Internet of Things, anche chiamato Internet of Everything, prevede la costituzione di una rete globale di macchine e dispositivi capaci di interagire autonomamente; questa rete rappresenta un sistema interconnesso che permette lo scambio di informazioni tra nodi. È definita come un'applicazione decentralizzata dell'IoT, dall'inglese Decentralized applications (Dapp). La tecnologia Blockchain permette la creazione, il coordinamento e la sincronizzazione di un complesso database distribuito, costituito da blocchi contenenti le transazioni avvenute tra i nodi di una rete. Questa è rappresentabile idealmente come una catena formata da blocchi contenenti gli eventi della rete. Le transazioni presenti nei blocchi devono essere validate dai nodi che compongono la rete, ciò dà luogo ad una rete di fiducia distribuita.

In altri termini, la Blockchain è un sistema di archiviazione dati sicuro, distribuito, ed immutabile condiviso tra una rete di attori. I dati vengono immagazzinati in "blocchi" (block), connessi l'uno all'altro in una catena (chain) tramite un hash, ovvero una funzione che converte caratteri alfanumerici in una nuova sequenza criptata e di lunghezza predeterminata.

Questi blocchi possiedono una "testa", che include metadati, e un corpo, che invece riguarda i dettagli dei dati veri e propri. Dato che ogni blocco è connesso al precedente e al successivo e distribuito tra tutti i partecipanti, al crescere del numero di attori nella rete diventa esponenzialmente più complesso modificare qualsiasi informazione. Esistono diversi tipi di Blockchain categorizzate a seconda del differente permesso di accesso. In altre parole, alcune Blockchain possono essere rese liberamente accessibili ("public" vs "private") e la capacità di scrivere sul registro illimitata o controllata ("permissionless" vs "permissioned"), ma esistono anche modelli più ibridi (come la Blockchain consortile).

Partita come un semplice modello per certificare i dati di un file digitale, nel tempo la Blockchain ha avuto numerose implementazioni. La prima e più aderente al modello originario riferisce al puro sistema di archiviazione dati. Garantendo l'immodificabilità dei dati registrati, la Blockchain viene utilizzata come strumento di verifica delle informazioni e impiegata nei più disparati sistemi di certificazione dei dati: a puro titolo esemplificativo come strumento di certificazione alimentare, registrazioni contratti e registro di proprietà di beni.

Tuttavia, il più noto utilizzo della tecnologia dopo il white paper di Satoshi Nakamoto del 2008 riferisce al suo impiego come tecnologia alla base della criptovaluta Bitcoin. Sfruttando la sicurezza dei dati la Blockchain consente di sviluppare un sistema di pagamento che funziona in assenza di una autorità centrale. Mentre infatti nei sistemi di e-money tradizionali quali le transazioni bancarie, il presupposto della correttezza dei dati deriva dalla fiducia nei sistemi centrali, nelle criptovalute la fiducia viene riposta nel sistema di archiviazione. Questo approccio noto come "zero knowledge proof space" fa in modo che soggetti che non hanno conoscenza reciproca possano tranquillamente scambiarsi criptovalute.

Il terzo impiego della Blockchain risale a innovazioni avvenute verso la metà degli anni '90 con l'introduzione degli Smart Contract. In sostanza uno Smart Contract altro non è che un programma per computer che registra e finalizza un accordo. Sebbene i suoi utilizzi siano tutt'altro che recenti (si pensi ai banali antivirus che rinnovano automaticamente la licenza allo scadere dei termini), la loro diffusione diviene massiccia grazie al lavoro di Vitalik Buterin, fondatore di Ethereum. Inseriti in un sistema di Blockchain, gli Smart Contract consentono la certificazione dei termini del contratto e la loro esecuzione automatica senza che via sia una terza parte coinvolta.

I problemi che le blockchain di terza generazione stanno cercando di risolvere sono legati alla scalabilità, in particolare attraverso la creazione di molteplici layer (tendenza che ha portato anche alla nascita di Lightning Network, layer di secondo livello per transazioni istantanee su Bitcoin), all'interoperabilità tra blockchain diverse e **allo sviluppo di tecnologie ad hoc per la realizzazione di applicazioni blockchain M2M (machine to machine) in ottica Internet of Things**.

La quarta applicazione della Blockchain si collega ai cosiddetti "Initial Coin Offering", innovativa modalità di crowdfunding basata sulla possibilità di creare nuovi modelli di business basati sulla tokenizzazione dei diritti. Il mercato complessivo delle ICO ha superato a settembre 2019 i 15 miliardi di euro.

La quinta e più recente implementazione della Blockchain riguarda il così detto **Web 3.0** noto anche come cripto internet. Il Web 3.0 nasce come contraltare allo strapotere GAFA (Google, Apple,

Facebook, Amazon) i cui modelli di business si basano sulla centralizzazione dei dati e del conseguente potere decisionale. Lavorando su modelli distribuiti, il Web 3.0 consente lo sviluppo di nuovi modelli di business capaci di sfruttare le risorse inutilizzate. Gli esempi non mancano. La startup Golem, che l'anno scorso ha sfiorato una capitalizzazione sul mercato delle ICO di quasi un miliardo di dollari, offre un modello distribuito alla capacità di calcolo che si contrappone ai modelli centralizzati di Amazon AWS. Riconoscendone le possibili implementazioni, in molti concordano che la Blockchain scatenerà la digital disruption di tutti i settori, facendo diventare distribuiti i modelli di business dominanti, creando valore da nuovi asset, riducendo i costi delle transazioni e incrementando la fiducia degli stakeholders. I primi settori a essere trasformati saranno quelli della finanza, dell'agroalimentare, della sanità, della moda, dello sport e intrattenimento, dei servizi professionali, della distribuzione e manifattura. La Blockchain pone anche molte sfide normative, in primis in merito alla tutela dei risparmiatori, e ambientali, in merito al consumo di energia richiesta dalla necessità di duplicare l'archiviazione dei dati.

Golem si appoggia all'algoritmo di consenso PoS (Proof of Stake), per cui i GNT non possono essere minati. **Il progetto si basa sul distributed computing, ovvero sul concetto di calcolo distribuito**. Esso non è altro che un sistema in cui tanti computer comunicano fra loro, pur essendo indipendenti. Chiunque può utilizzare Golem per far girare diversi tipi di software, in diversi settori tra i quali computer grafica, business, machine learning, crittografia, ecc… Cos'è che si condivide tramite questo network decentralizzato su blockchain Ethereum? La potenza di calcolo del proprio pc. Essa viene ceduta dagli utenti al network stesso. In poche parole si va a prestare una parte della potenza fornita dal PC per essere pagati con il cripto valuta Golem. Si può usare Golem per fare numerose attività, tra cui le predizioni di mercato, inoltre è consentito sviluppare e vendere propri software sul network di Golem. Il network si basa sulla rete peer-to-peer ed usa un linguaggio open source.

In breve possiamo dire che l'infrastruttura elementare di una blockchain è costituita da:

- Database distribuito
- Meccanismo di consenso
- Token come premio di convalida.

Il funzionamento della blockchain include inoltre ulteriori componenti come:

- Nodo
- Transazione
- Blocco

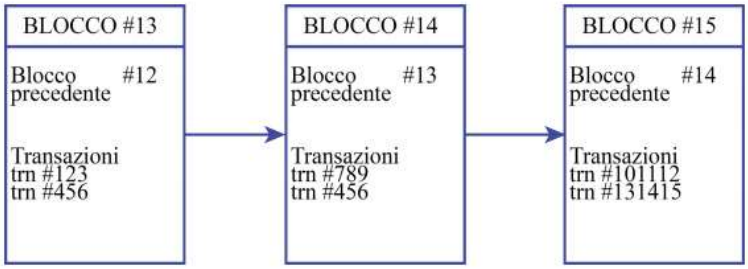
• Ledger (registro)

• Hash

• Miners

Riepilogando, ogni blocco della catena può contenere un certo numero di transazioni. Le transazioni riguardano lo scambio di risorse digitali, ed utilizzano una rete peer-to-peer che memorizza queste transazioni in maniera distribuita attraverso la rete.

Gli attori proprietari dei beni digitali e le transazioni che comportano un cambio di proprietà sono registrati all'interno del blocco mediante l'utilizzo della crittografia a chiave pubblica/ privata e delle firme digitali che garantiscono sicurezza e autenticità allo scambio. Ogni blocco possiede un valore di **hash** identificativo. L'hash è in grado di mappare una stringa numerica o di testo, in una stringa unica ed univoca di lunghezza determinata. In questa maniera grazie all'hash, si è in grado di identificare in maniera univoca e sicura ciascun blocco. L'hash è strutturato in modo da impedire la rievocazione del testo o la stringa numerica da cui esso è stato generato. Inoltre ogni blocco oltre ad avere il proprio hash identificativo contiene anche l'hash del blocco che lo precede. In questa maniera, quando un nuovo blocco viene aggiunto alla catena di blocchi, questo mantiene una visione condivisa e concordata dello stato attuale della blockchain. Un esempio è presentato nella figura successiva.



Il **ledger** (registro) contiene lo stato condiviso e concordato della catena di blocchi e l'elenco di tutte le transazioni che sono state elaborate. In tale maniera, tutti i nodi che partecipano alla rete di questo sistema decentralizzato avranno una copia dell'intera catena di blocchi che viene continuamente aggiornata e sincronizzata tra tutti i nodi della rete. Questo aspetto è fondamentale per la tecnologia blockchain, perché in questa maniera non esiste un punto centrale di vulnerabilità che gli hacker possono sfruttare, come invece può accadere per i database centralizzati.

Nel caso in cui qualcuno fosse intenzionato a modificare qualche transazione all'interno di un blocco, questo modificherebbe il valore hash identificativo e quindi affinché l'attacco possa andare a buon fine, la modifica deve essere replicata a sua volta su tutti i nodi della rete. Questa operazione

richiederebbe una potenza di calcolo enorme che, con le tecnologie attualmente esistenti, risulterebbe impossibile.

L'architettura peer-to-peer contribuisce alla sicurezza e all'immutabilità delle transazioni e dei blocchi registrati nella blockchain. Colui che convalida le transazioni all'interno di un blocco e aggiunge il blocco alla catena prende il nome di **miner**. Il miner convalida il blocco attraverso un meccanismo di consenso, che corrisponde alla risoluzione di un complesso problema matematico. Nel caso specifico della blockchain Bitcoin questo sforzo computazionale che comporta un importante consumo di energia elettrica, prende il nome di "**proof of work**". L'intera sicurezza e validità della catena è garantita proprio dal lavoro dei miners. Il ruolo del miner è fondamentale per il corretto funzionamento della blockchain. Il miner è un utente volontario che partecipa liberamente all'interno della rete mettendo a disposizione la propria CPU (Central Processing Unit) per risolvere questi problemi matematici che permettono di convalidare i blocchi. In breve, il compito del miner è quello di raggruppare e verificare le transazioni che ancora non sono state inserite all'interno di un blocco e dopo averle verificate, provare a risolvere il problema computazionale svolgendo tale proof of work.

Quando è stata trovata la soluzione a tale problema, il blocco viene trasmesso alla rete e dopodiché avviene l'aggiornamento della catena per tutti i nodi della rete. Un aspetto molto importante è che il miner che risolve e quindi conseguentemente convalida il blocco, deve aggiungere il nuovo blocco alla catena più lunga esistente, questo è un passaggio fondamentale per la salvaguardia dell'intera piattaforma. Nella blockchain Bitcoin, un nuovo blocco viene aggiunto alla catena dopo circa 10 minuti. È importante sottolineare come oltre alla proof of work esistono anche altri meccanismi di consenso come ad esempio la **proof of stake**. La proof of stake semplifica il processo relativo al mining descritto precedentemente. Per quanto riguarda la proof of stake, il lavoro richiesto per eseguire il processo di verifica viene ripartito tra i singoli membri in base alla loro percentuale di partecipazione. Ad esempio, se un utente possiede il 20% del totale delle attività di blockchain in circolazione, l'utente dovrà eseguire il 20% dell'attività di mining richiesta. In questa maniera si riduce la complessità del processo di verifica decentralizzata e si possono quindi generare anche dei risparmi relativi ai costi energetici e operativi (Hasse et al., 2016).

Siccome il compito del miner è di assoluta importanza per mantenere la sicurezza dell'intera catena di blocchi, il suo sforzo viene remunerato attraverso un **token**. L'incentivo che viene dato ai miners attraverso il token per la risoluzione del problema è la chiave principale affinché l'intero sistema sia completamente affidabile. Nella blockchain Bitcoin, il miner per il suo sforzo ottiene appunto dei Bitcoin. Nel 2009 per ogni blocco validato il miner riceveva in cambio 50 Bitcoin. Tale valore, per

come è stato strutturato l'algoritmo, viene dimezzato ogni quattro anni; ad oggi per ogni blocco validato il sistema riconosce come compenso 12,5 Bitcoin. Ovviamente questo vale esclusivamente per la blockchain Bitcoin. Infatti ogni blockchain si struttura intorno ad un algoritmo che avrà regole differenti che dipendono dalle logiche di programmazione e degli obiettivi e funzionalità offerte dal sistema stesso.

Il token in una blockchain pubblica ricopre un ruolo molto importante. **Esso può essere identificato come un insieme di informazioni digitali capace di attribuire il diritto di proprietà ad un determinato soggetto**. Il token consiste in un insieme di informazioni registrate sulla blockchain che attraverso un protocollo possono essere trasferite. Il token più "famoso" è appunto il Bitcoin ma, dopo di esso, ne sono comparsi molti altri. A tal proposito un esempio è l'Ether che è il token appartenente alla blockchain pubblica Ethereum. Al momento possiamo distinguere tre tipologie di token:

- i token di classe 1 che rappresentano una vera e propria moneta e che tramite la blockchain possono essere trasferiti (es: Bitcoin);
- token di classe 2 che permettono di esercitare alcuni diritti verso una controparte;
- i token di classe 3 che hanno un ruolo misto, ovvero che raffigurano diritti di comproprietà ed alla stessa maniera attribuiscono diritti diversi come per esempio il diritto di voto.

Inizialmente si è parlato di **chiavi crittografiche**: vediamone ora il funzionamento in relazione alla blockchain Bitcoin. Nello specifico il sistema Bitcoin si basa su due tecnologie crittografiche: crittografia a chiave pubblico-privata e la crittografia per le transazioni di rete. Come spiegato precedentemente ad ogni transazione è correlata una firma digitale che è diversa per ogni transazione. La tecnologia che permette tutto questo è la crittografia a chiave pubblico-privata, che permette con la chiave privata di creare una "firma" associata ad una chiave pubblica. La chiave pubblica è condivisa nella rete, mentre la chiave privata è personale ed è utilizzata per de-crittografare i dati. Inoltre di fondamentale importanza è la "**crittografia ellittica**" che praticamente permette di calcolare la chiave pubblica data la chiave privata ma non permette il contrario. In questa maniera tutti gli utenti che partecipano alla rete sono identificabili attraverso la loro chiave pubblica. Per tale motivo nessun ulteriore dato personale è disponibile all'interno della rete. Tutto questo permette l'anonimato degli utenti o meglio, come sostengono alcuni autori (Swan, 2015) che le transazioni non siano realmente anonime ma "pseudo anonime".

In generale si può riassumere che ci sono tre tipologie di blockchain:

• **Blockchain pubblica**. La blockchain pubblica è una blockchain nella quale chiunque può diventare un nodo della rete, chiunque può leggere e inviare transazioni che poi saranno

successivamente incluse e validate in un blocco, chiunque può essere un miner e per tale motivo partecipare al meccanismo di consenso offrendo volontariamente la propria potenza di calcolo. Come già definito in precedenza, la blockchain pubblica è sicura grazie alla presenza di un incentivo economico che ripaga lo sforzo compiuto dai miner, grazie alla crittografia ed dal principio secondo il quale il grado di influenza di un singolo attore all'interno della rete nel processo di consenso, è proporzionale alla quantità di risorse economiche che può apportare. Questa tipologia di blockchain è definita "completamente decentralizzata".

• **Blockchain privata**. La blockchain privata è una blockchain in cui le autorizzazioni di scrittura all'interno dei blocchi sono mantenute completamente centralizzate. Per quanto riguarda invece le autorizzazioni di lettura della blockchain, queste possono essere pubbliche o anch'esse limitate ad un numero finito di utenti. In questa maniera una blockchain privata è sicuramente più vicina ai modelli di business più tradizionali, nonostante questo non debba necessariamente essere visto come un aspetto negativo. Il fatto che questa tipologia di infrastruttura, almeno a prima vista, non abbia lo stesso impatto rivoluzionario della blockchain pubblica, non significa che non possa comunque svolgere un ruolo preponderante nel processo di efficientamento di un'attività di business.

• **Permissioned Blockchain (Consortium)**. La permissioned blockchain a differenza delle precedenti è una blockchain in cui il meccanismo di consenso è controllato da un insieme di nodi preselezionati. Si pensi ad un "consorzio" di 10 istituti finanziari, ognuno dei quali gestisce un nodo. In questo caso è sufficiente che 8 di loro firmino un blocco affinché il blocco sia valido. A tal proposito il diritto di leggere la blockchain può essere sia pubblico che limitato ad alcuni partecipanti. Questa tipologia di blockchain è definita "parzialmente decentrata".

A prima vista potrebbe non essere molto chiara la differenza tra una blockchain privata ed una permissioned, per quanto riguarda la blockchain, essa è fondamentalmente un "ibrido" tra la "bassa fiducia" (minor controllo) che fornirebbe la blockchain pubblica e "la singola entità altamente affidabile" che invece contraddistingue la blockchain privata. La blockchain privata può essere definita come un sistema centralizzato tradizionale con l'aggiunta di un grado di verificabilità crittografica.

Per quanto riguarda le blockchain private si possono individuare tali vantaggi:

- Sia che si tratti di un consorzio o di una blockchain completamente privata, nel caso in cui fosse necessario, sarebbe possibile in maniera più semplice modificare le regole della piattaforma blockchain, o per esempio ripristinare delle transazioni.
- Chi svolge il ruolo del miner è noto a priori e gode di una fiducia pregressa.
- Le transazioni sono convalidate in maniera più veloce, perché solo alcuni nodi hanno questo ruolo.
- I possibili errori possono essere risolti in breve tempo attraverso un intervento manuale.
- Se i permessi di lettura all'interno di una blockchain privata sono limitati, si ottiene una privacy maggiore sui dati.

Alla luce di queste considerazioni, può sembrare che in realtà le blockchain private siano la scelta migliore per una istituzione, come può essere la pubblica amministrazione. In realtà, anche in un contesto come quello del settore pubblico, la blockchain pubblica ha sempre il suo grande valore e questo valore risiede soprattutto nelle virtù filosofiche dei suoi principali sostenitori che promuovono la libertà, la neutralità e l'apertura.

A tal proposito i vantaggi di una blockchain pubblica possono essere suddivisi in due grandi categorie:

- La blockchain pubblica fornisce un modello di protezione riguardante gli utenti di una certa applicazione dagli stessi sviluppatori di quell'applicazione. In questa maniera ci sono alcune cose che neanche gli stessi sviluppatori possono essere in grado di fare. Questo meccanismo permette di rendere molto difficile se non impossibile la possibilità di effettuare dei cambiamenti alla catena, garantendo una maggiore fiducia degli utenti nei confronti del sistema.
- La blockchain pubblica è aperta, immutabile e può essere utilizzata da tutti e letta da tutti ma allo stesso tempo garantisce agli utenti l'anonimato (o lo pseudo-anonimato) all'interno della rete. Questo crea un effetto rete all'interno della piattaforma che rende la blockchain sempre più sicura e protetta. Di contro a questa estrema sicurezza vi è la lentezza nella validazione nei blocchi (in Bitcoin ogni 10 minuti) e il dispendioso spreco energetico dovuto al lavoro compiuto dai miners.

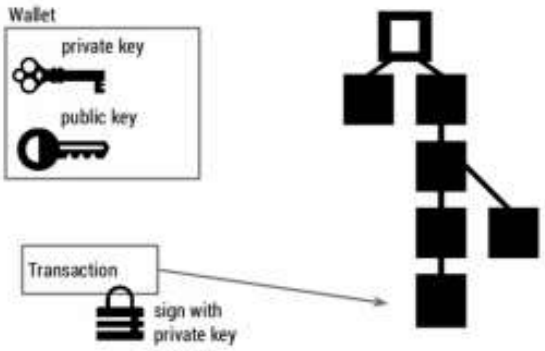
Alla luce di questa analisi è facile capire che la situazione ottimale, in termini di implementazione della tecnologia, varia da settore a settore. In alcuni casi l'implementazione di una blockchain

pubblica può essere chiaramente la scelta ottimale, in altri casi invece, il maggior controllo dato dalla blockchain privata la rende necessaria per un certo sistema. Si pensi per esempio al settore pubblico, dove la fiducia nel sistema può essere pregressa. Per tale motivo una blockchain privata (o permissioned) potrebbe essere preferita rispetto ad una pubblica. Per tutte queste ragioni la risposta riguardante la scelta migliore tra le due è ovviamente: dipende.

Ethereum

Ethereum è una piattaforma decentralizzata, pubblica e open-source basata su blockchain e dotata di funzionalità di creazione e pubblicazione di contratti intelligenti scritti in un linguaggio di programmazione Turing completo. Proposto inizialmente da Vitalik Buterin nel 2013, il suo sviluppo è iniziato l'anno seguente e oggi è principalmente curato dall'Ethereum Foundation.

Ethereum dispone di una cripto valuta nativa, l'Ether, che oltre a poter essere scambiato fra account, è generato dalla piattaforma stessa come ricompensa ai miner per il lavoro computazionale svolto. A ciascun account (o wallet) sono associati una chiave privata e una chiave pubblica che prende anche il nome di indirizzo. Per poter effettuare una transazione è necessario che il mittente conosca l'indirizzo, ovvero la chiave pubblica, del wallet del destinatario e che firmi digitalmente la transazione prima di inviarla alla rete Ethereum con la propria chiave privata, dimostrando così che il richiedente dell'operazione è l'effettivo titolare del wallet.



Ethereum mette a disposizione anche l'Ethereum Virtual Machine (EVM), una macchina virtuale decentralizzata all'interno della quale vengono eseguiti gli Smart Contract utilizzando la potenza computazionale dei nodi che costituiscono la rete. Gli Smart Contract possono essere scritti utilizzando diversi linguaggi di programmazione, il più utilizzato dei quali è al momento Solidity. Il codice sorgente scritto in Solidity, o in un qualsiasi altro linguaggio ad alto livello fra quelli supportati, deve essere poi compilato per produrre un bytecode pronto per essere eseguito dalla EVM.

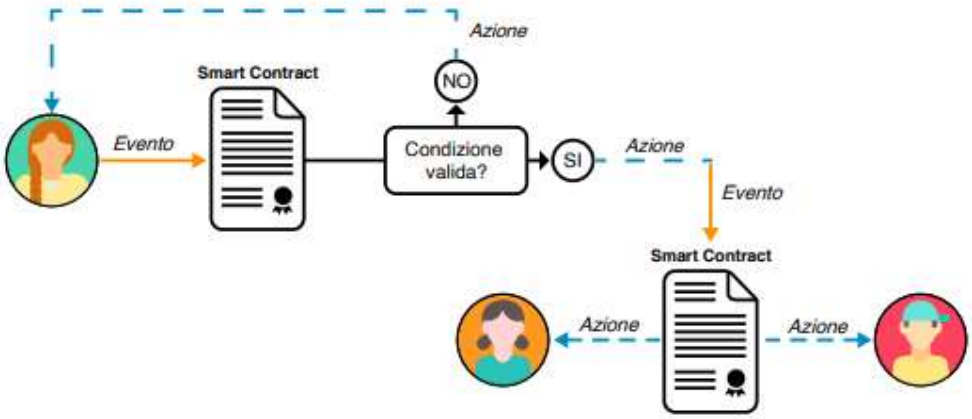
Ciascuna transazione generata sulla rete Ethereum, sia essa un semplice trasferimento di Ether, o l'invocazione di una funzione di uno smart contract, comporta un costo che è proporzionale alla complessità computazionale, alla banda utilizzata e/o alla quantità di storage necessario. Questo meccanismo interno adottato da Ethereum prende il nome di "gas", ed **è stato ideato per evitare possibili attacchi spam o DDoS (Distributed Denial of Service)** che potrebbero compromettere il funzionamento dell'intera rete. Per ogni transazione l'utente può dunque definire un gas price e un gas limit, ovvero rispettivamente l'ammontare che decide di pagare per ogni unità di gas consumata (valore espresso solitamente in Gwei, un sottomultiplo del l'Ether) e la quantità massima di gas consumabile. In questo modo il mittente della transazione conosce a priori quale sarà il costo massimo dell'operazione, e garantisce al tempo stesso che ogni computazione giungerà sempre a termine.

Qualora infatti si verificasse ad esempio un loop infinito all'interno del codice di uno smart contract, la computazione verrà interrotta non appena si sarà consumato tutto il gas messo a disposizione al momento della creazione della transazione. Il costo effettivo sarà determinato dall'effettiva quantità di gas utilizzato moltiplicato per il gas price indicato e quindi detratto al mittente dal saldo del suo wallet in Ether. La possibilità di specificare il gas price fa sì che sia **possibile assegnare una priorità alle transazioni**, i miner preferiranno infatti inserire all'interno del prossimo blocco le transazioni con un gas price più elevato, per ottenere una maggiore ricompensa. Mediamente **la rete Ethereum genera un nuovo blocco ogni 14/15 secondi**, un tempo molto più breve rispetto a quello impiegato da **Bitcoin che si aggira intorno ai 10 minuti**. Per la messa in sicurezza della rete e per il conseguimento del consenso distribuito, Ethereum, come Bitcoin, basa il suo funzionamento sull'utilizzo di un protocollo Proof-of-work (PoW).

Gli Smart-Contract

La tecnologia blockchain grazie alle sue caratteristiche peculiari ha dato vita a una nuova forma di contratto, chiamato Smart Contract. Il concetto di Smart Contract è stato introdotto per la prima volta alla fine degli anni '90 da Nick Szabo ma, è rimasto un concetto piuttosto astratto fino al lancio di Ethereum nel 2015. In breve**, uno Smart Contract non è altro che un programma, memorizzato su una blockchain e che esegue una transazione nel momento in cui si verifica una determinata condizione**. Per tale motivo lo Smart Contract è in generale una dichiarazione come "trasferimento da A a B se si verifica Y".

Quindi si può dire che uno Smart Contract può essere visto simile al servizio IFTTT (if this then that) la quale produce gli effetti contenuti in una transazione al verificarsi di eventi specifici:



Attualmente il contratto nel senso tradizionale è un accordo tra due o più parti in cui si dichiara di fare o non fare qualcosa in cambio di qualcos'altro (Swan, 2015). Per tale motivo ciascuna delle parti deve fidarsi dell'altra per assolvere al proprio obbligo. Lo Smart Contract rappresenta in forma digitale lo stesso contratto, ma permette l'eliminazione della necessità di un rapporto fiducia tra le parti. "Questo perché un contratto intelligente è sia definito dal codice che eseguito (o applicato) dal codice, automaticamente senza discrezione" (Swan, 2015).

Lo Smart Contract è costituito da tre elementi: l'autonomia, l'autosufficienza e il decentramento. Per quanto riguarda **l'autonomia**, questo significa che una volta che il contratto è "sottoscritto", i contraenti possono anche non essere più in contatto; con **autosufficienza** si intende la capacità di uno Smart Contract di gestire le risorse, ovvero raccogliere fondi fornendo servizi o emettere azioni (Swan, 2015).

Infine gli Smart Contract sono **decentralizzati**, distribuiti e automatizzati attraverso i nodi della rete. La grande promessa che lo Smart Contract rappresenta è la possibilità dell'evolversi di un nuovo ecosistema di automazione tecnica per produrre un nuovo tessuto sociale che permetta efficienze civiche, mobilità personale e trasformazione istituzionale. In questa maniera, lo Smart Contract rappresenta una visione automatizzata del futuro.

Ma, come ha dimostrato Turing, non si può prevedere se un contratto Smart terminerà, o per quanto tempo durerà, senza che venga effettivamente eseguito. Naturalmente, tra un programma che richiede un millisecondo per essere validato e uno che funziona per sempre, c'è un'infinita gamma di programmi cattivi, che monopolizzano le risorse, che fanno fluttuare la memoria, che surriscaldano la CPU e che quindi sprecano risorse. Come si fà a limitare le risorse utilizzate da un contratto intelligente se non è in grado di prevedere in anticipo la loro quantità necessaria?

Per rispondere a questa sfida, Ethereum introduce un meccanismo di misurazione **chiamato gas**. Poiché l'EVM (Ethereum Virtual Machine) che esegue un contratto intelligente, tiene conto accuratamente di ogni istruzione (calcolo, accesso ai dati, ecc.). Ogni istruzione ha un costo predeterminato in unità di gas. Quando una transazione fa scattare l'esecuzione di un contratto smart, deve includere una quantità di gas che stabilisce il limite superiore di calcolo che può essere consumato in esecuzione del contratto Smart. L'EVM terminerà l'esecuzione se la quantità di gas consumata dal calcolo supera il gas disponibile nella transazione. Il gas è il meccanismo che Ethereum utilizza per consentire il calcolo, limitando al contempo le risorse che qualsiasi programma può consumare.

Ethereum e Web 3.0

Il progetto Ethereum non concerne soltanto lo sviluppo della nota blockchain, il fondatore Vitalik Buterin ha infatti obiettivi molto più ambiziosi. La roadmap di Ethereum prevede infatti lo sviluppo di altre tecnologie complementari che si andranno ad affiancare alla blockchain per realizzare quello che secondo l'Ethereum Foundation diverrà il Web 3.0, ovvero la prossima generazione del Web.

I contratti intelligenti nati da un'idea semplice, avere una serie di istruzioni auto-eseguibili tra due parti che non devono essere supervisionate o imposte da una terza parte, hanno permesso a Ethereum di creare un ambiente in cui gli sviluppatori di tutto il mondo potevano creare, a partire da questi, la propria applicazione decentralizzata , nota anche come Dapp.

Se con Web 1.0 ci si riferisce al primo stadio del World Wide Web, costituito da pagine web statiche, connesse tra loro da semplici hyperlink, e da prime semplici applicazioni come blog e forum, la transizione al Web 2.0 si è avuta con l'avvento di AJAX e altre tecnologie che hanno reso il web sempre più interattivo e dinamico, consentendo agli sviluppatori di realizzare web application sempre più complesse, dove gli utenti possono interagire e modificare i contenuti delle pagine web. **Secondo la visione del progetto Ethereum la caratteristica principale della prossima generazione del World Wide Web sarà la decentralizzazione, un web senza più alcun server centralizzato e fatto di dApps**.

Qualsiasi applicazione non banale per offrire i suoi servizi richiede però tipicamente diverse risorse per poter essere eseguita: potenza computazionale, un database, memoria di massa e un sistema di comunicazione per poter interagire con altre applicazioni.

Come discusso nelle precedenti sezioni, blockchain come quella di Ethereum sono in grado di offrire solo i primi due tipi di queste risorse: **la EVM fornisce potenza computazionale** per eseguire il

codice contenuto negli Smart Contract che realizzano il business logic delle applicazioni decentralizzate, mentre **la blockchain stessa può memorizzare dati e transazioni esattamente come un database**. Tuttavia ciò non è sufficiente, infatti per poter sviluppare applicazioni complesse e totalmente decentralizzate è necessario disporre anche di **una memoria di massa** dove poter memorizzare file dati, anche di grandi dimensioni, e di **un sistema di comunicazione**.

A tal scopo il progetto mira a fornire:

- **Smart Contract**: logica decentralizzata
- **Swarm:** storage decentralizzato
- **Whisper**: messaggistica decentralizzata

L'insieme di queste tecnologie può consentire lo sviluppo di applicazioni totalmente decentralizzate e arbitrariamente complesse, andando a completare la visione di Ethereum come metafora di un computer condiviso, globale e decentralizzato.

Whisper

Whisper è un protocollo di messaggistica peer-to-peer per applicazioni decentralizzate che fornisce agli sviluppatori di dApps delle semplici API per poter inviare e ricevere messaggi. Le comunicazioni avvengono off-chain, quindi Whisper è un protocollo totalmente indipendente dalla blockchain. La caratteristica principale di Whisper è la cosiddetta **darkness**, ovvero la possibilità di scambiare messaggi quasi in totale segretezza. Whisper non si limita a garantire la confidenzialità sul contenuto dei messaggi mediante crittografia, ma nasconde anche l'identità del mittente e del destinatario rendendo impossibile agli altri nodi della rete sapere se due entità stanno intrattenendo tra loro una conversazione. Questa scelta ha costretto i progettisti di Whisper a fare alcuni sacrifici in termini di performance per tutelare maggiormente la privacy. Per questo motivo non è un protocollo adatto ad ogni caso d'uso, in particolare non è indicato per comunicazioni in tempo reale e per l'invio di blocchi di dati di grandi dimensioni. Al contrario questo protocollo è stato progettato per inviare piccole quantità di informazioni non persistenti tra Dapp o sviluppare ad esempio **app di messaggistica focalizzate sulla privacy**. Whisper è in ogni caso un protocollo altamente configurabile e consente agli sviluppatori una grande flessibilità nel controllo dei parametri di sicurezza e privacy dei loro messaggi. **Alla base del funzionamento di Whisper vi è una rete decentralizzata di computer chiamati nodi**.

Per costituire questa rete, un nodo trova i suoi pari utilizzando un algoritmo di node-discovery simile a quello utilizzato da BitTorrent per la ricerca dei peer. Whisper è incluso di default all'interno dei principali client Ethereum, Geth e Parity, ma è attualmente configurato come feature opzionale da attivare manualmente se lo si desidera. L'utilizzo sulla rete principale è quindi limitato dal numero di nodi Ethereum in esecuzione che hanno il protocollo Whisper abilitato. Ogni nodo connesso alla rete Whisper dispone di un'identità. Un'identità in Whisper è un'entità (un individuo o un gruppo) titolare di una chiave crittografica che consuma messaggi.

Senza una chiave crittografica è impossibile inviare e ricevere messaggi. A seconda del caso d'uso Whisper supporta sia la crittografia simmetrica (AES-256) che quella asimmettrica (SECP-256k1). La crittografia garantisce che solo gli effettivi destinatari possano accedere al contenuto di un messaggio. Se un nodo può decodificare un messaggio, allora questo è destinato a un utilizzatore di quel nodo. Come anticipato Whisper rende possibile la comunicazione tra due peer in totale riservatezza, senza lasciare alcuna prova né ad eventuali analizzatori del traffico né agli altri peer, anche se questi hanno partecipato all'instradamento dei messaggi. Per realizzare ciò, il meccanismo di scambio dei messaggi risulta essere piuttosto complesso. Innanzitutto per poter inviare un messaggio questo va prima crittografato utilizzando una chiave simmetrica condivisa o la chiave pubblica del destinatario.

È impossibile inviare messaggi non crittografati tramite Whisper. Se lo desidera, il mittente può allegare al messaggio anche una firma digitale che garantirà al destinatario l'identità del mittente. La firma, qualora fornita, è la signature ECDSA dell'hash Keccak-256 dei dati non crittografati ottenuta utilizzando la chiave privata del mittente. Una volta crittografato il messaggio questo viene inserito all'interno di un pacchetto che prende il nome di envelope. Questo aggiunge una serie di metadati necessari per l'instradamento del messaggio. Contrariamente però a quanto avviene nella maggior parte dei protocolli, gli envelope di Whisper non contengono alcuna informazione sul destinatario.

Il formato di un envelope è quindi tipicamente il seguente:

[Version, Expiry, TTL, Topic, AESNonce, Data, EnvNonce]

Whisper per sconfiggere l'analisi del traffico e per recapitare il messaggio a destinazione senza conoscere l'identità del destinatario fa sì che ogni messaggio sia instradato su ogni nodo della rete. In questo senso, Whisper si comporta come il protocollo UDP (User Datagram Protocol) quando **opera in modalità broadcast**. Ogni nodo inoltra dunque il messaggio ai nodi vicini fino a quando il destinatario non lo riceve. Il destinatario sarà l'unico in grado di decifrare il messaggio poiché l'unico a disporre della chiave crittografica necessaria.

In ogni caso **anche il nodo destinatario effettuerà l'inoltro del messaggio ai suoi nodi vicini per essere completamente irrintracciabile**. Poiché in questo modo ciascun nodo dovrebbe tentare di decifrare ogni messaggio che transita sulla rete per scoprire se era a lui il destinatario della comunicazione, e dal momento che la decrittazione è un lavoro computazionalmente molto costoso, i progettisti di Whisper hanno risolto questo problema richiedendo che a ciascun messaggio sia associato un argomento da memorizzare all'interno del campo Topic. Più precisamente gli argomenti dei messaggi vengono automaticamente sottoposti a hash SHA3-256 e solo i primi 4 byte del risultato vengono memorizzati all'interno dell'envelope. Ciascuna identità registra gli argomenti a cui è interessata e utilizzando la sua chiave crittografica crea un filtro per i messaggi su un nodo. Quest'efficiente filtro probabilistico, noto come bloom filter, può indicare con un grado molto elevato di certezza se un messaggio appartiene a uno degli argomenti di interesse. Il nodo tenterà la decrittazione solo se un filtro segnala una possibile corrispondenza. In questo modo **grazie all'utilizzo del pattern di comunicazione publish–subscribe, Whisper diventa un protocollo di comunicazione estremamente semplice ed efficiente.**

Whisper adotta infine un protocollo proof-of-work per contrastare **possibili attacchi di tipo Denial of Service (DoS)**. Senza questa misura sarebbe infatti possibile attaccare la rete inviando una raffica di messaggi a tutti i nodi (**flood attack**), o messaggi con un TTL (Time-to-Live espresso in secondi) molto elevato che ne causerebbe una persistenza estremamente duratura all'interno della rete (**expiry attack**). Per questo motivo è stato previsto all'interno dell'envelope un campo chiamato EnvNonce all'interno del quale viene memorizzato il risultato della proof-of-work.

La PoW di Whisper consiste nell'esecuzione ripetuta di un semplice algoritmo SHA3 con cui si cerca di trovare il numero più piccolo possibile entro un determinato periodo di tempo. Il mittente del messaggio può decidere quanto tempo dedicare a questa operazione attraverso la specificazione di un parametro e tanto più lavoro si sceglie di eseguire localmente, tanto più a lungo verrà mantenuto il messaggio presso i vari nodi e tanto più velocemente questo si propagherà attraverso la rete, acquisendo una priorità maggiore rispetto agli altri messaggi.

Infatti ad ognuno di essi viene assegnato un punteggio secondo i seguenti criteri:

• i messaggi più piccoli hanno voti più alti;

• i messaggi con PoW più alta hanno punteggi più alti;

• i messaggi con TTL inferiore hanno punteggi più alti. Sulla base di questo punteggio ciascun nodo elaborerà i messaggi (e li inoltrerà ai nodi vicini) solo se questo valore supererà una certa soglia, altrimenti verranno eliminati.

IPFS

IPFS (InterPlanetary File System) è un protocollo e un file system peer-to-peer distribuito pensato per la memorizzazione e la condivisione di contenuti ipermediali. IPFS fornisce un modello di archiviazione a blocchi ad alto throughput e una localizzazione delle risorse basata su collegamenti ipertestuali indirizzati al contenuto. IPFS è stato ideato con l'ambizioso obbiettivo di voler essere la base per la creazione di un nuovo Web completamente distribuito, più veloce, più sicuro e più aperto, ponendosi come alternativa all'attuale architettura client-server e al celeberrimo protocollo HTTP.

I vantaggi che si potrebbero trarre dall'utilizzo di IPFS sarebbero molteplici, ad esempio:

• **download più veloci ed efficienti**: mentre HTTP può scaricare una risorsa da un solo computer alla volta, IPFS può recuperare parti diverse di un file da molteplici computer contemporaneamente;

• **un web più robusto**: l'architettura di IPFS elimina i server centrali che rappresentano un single point of failure permettendo la creazione di un Web "permanente". IPFS crea una rete resiliente dove ogni risorsa è sempre disponibile grazie al mirroring dei dati su più nodi;

• **resistenza alla censura**: mentre può essere molto semplice per un governo bloccare l'accesso a un determinato sito internet ospitato presso un server centralizzato, IPFS crea un Web immune alla censura.

IPFS basa il suo funzionamento sull'utilizzo di molteplici tecnologie, racchiudendo in sé le idee di successo che erano alla base di precedenti sistemi peer-to-peer, inclusi DHT, BitTorrent, Git e SFS. Per cercare di spiegare come funziona IPFS, il primo concetto da introdurre è quello di indirizzamento basato sul contenuto, che si contrappone al paradigma di indirizzamento basato sulla posizione utilizzato oggigiorno nel Web.

All'interno del World Wide Web infatti, le risorse sono sempre state identificate per mezzo di un URL, che esprime in modo esatto dove è localizzata la risorsa desiderata, specificando l'host a cui è necessario connettersi (mediante dominio o indirizzo IP) e il path per l'individuazione del file all'interno del file system. Questo approccio ha il chiaro svantaggio che qualora il server non risulti raggiungibile per un qualsiasi motivo, la risorsa desiderata non sarà disponibile nonostante è estremamente probabile che almeno un altro computer nel mondo ne stia mantenendo una copia. IPFS supera questo problema identificando le risorse, non mediante la propria posizione, bensì attraverso il loro contenuto. Ogni file in IPFS è infatti identificato attraverso un hash univoco che viene calcolato sul contenuto del file stesso.

Qualora un peer desideri scaricare una determinata risorsa, sarà sufficiente che conosca l'hash di quel file e attraverso un'interrogazione della rete otterrà risposta da uno o più peer che detengono una copia di quella risorsa. L'utilizzo dell'hash come chiave per l'identificazione dei file ha inoltre il vantaggio di garantirne l'integrità: il nodo che ha richiesto una risorsa, per assicurarsi che il file ottenuto sia effettivamente quello richiesto e che non abbia subito alcuna alterazione, non dovrà far altro che ricalcolarne l'hash e verificare che questo corrisponda con quello che aveva inizialmente richiesto. Oltre ad aggiungere questa misura di sicurezza gratuitamente, la scelta di utilizzare l'hash come chiave porta un secondo vantaggio, ovvero la deduplicazione dei file: ogni volta che un utente pubblica un nuovo file su IPFS, la rete verifica attraverso il suo hash se questo è già presente, evitando automaticamente che ne vengano mantenute molteplici copie qualora più utenti caricassero la medesima risorsa.

Dapp e web3.0

La creazione di Dapp è una delle funzionalità più importanti di Ethereum. Oltre ad essere decentralizzata, ci sono alcune altre caratteristiche che deve avere una Dapp:

- Il codice sorgente della Dapp dovrebbe essere aperto a tutti
- L'applicazione deve avere una sorta di token per alimentarsi
- L'App deve essere in grado di generare i propri token e avere un meccanismo di consenso integrato

Tutto ciò è reso possibile codificando gli Smart Contract usando Solidity , un linguaggio volutamente snellito e con una sintassi molto simile a ECMAScript (Javascript). Oltre a creare il contratto intelligente, deve essere reso disporre di un ambiente in cui eseguirlo. Vediamo quindi alcune proprietà che questo ambiente di esecuzione deve avere.

Un ambiente di esecuzione per Dapp deve essere:

- Deterministico
- Terminabile
- Isolato

Proprietà n. 1: deterministico

Un programma è deterministico se fornisce ogni volta lo stesso output a un dato input. Per esempio. Se 3+1 = 4 allora 3+1 sarà SEMPRE 4 (assumendo la stessa base). Quindi, quando un

programma fornisce lo stesso output allo stesso insieme di input in computer diversi, il programma viene chiamato deterministico. L'ambiente deve assicurarsi che l'esecuzione del contratto intelligente sia sempre deterministica.

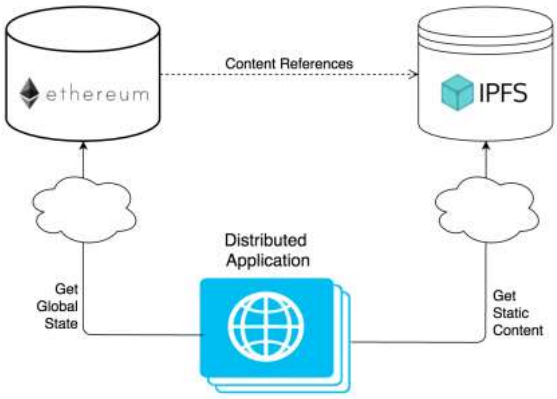
Proprietà n. 2: Terminabile

In logica matematica, abbiamo un errore chiamato "problema di arresto". Fondamentalmente, afferma che non è possibile sapere se un determinato programma può eseguire o meno la sua funzione entro un limite di tempo. Nel 1936, Alan Turing dedusse, usando il problema diagonale di Cantor, che non c'è modo di sapere se un dato programma può finire in un tempo limite o meno. Questo è ovviamente un problema per i contratti intelligenti perché, per definizione, i contratti devono poter essere rescissi in un determinato limite di tempo. Quindi l'ambiente deve essere in grado di fermare il funzionamento dello Smart Contract.

Proprietà n. 3: Isolato

In una blockchain , chiunque può caricare uno Smart Contract. Tuttavia, a causa di ciò, i contratti possono, consapevolmente e inconsapevolmente, contenere virus e bug. Se il contratto non è isolato, ciò può ostacolare l'intero sistema. Pertanto, è fondamentale mantenere un contratto isolato in una sandbox per salvare l'intero ambiente da eventuali effetti negativi. Ethereum esegue i suoi contratti intelligenti utilizzando una macchina virtuale chiamata Ethereum Virtual Machine (EVM).

Un altro requisito richiesto è che l'applicazione sia totalmente decentralizzata, e che quindi non utilizzi nessun server centrale, nemmeno per l'hosting del sito web per il rendering del processo di elaborazione. Per risolvere questo problema si è pensato di utilizzare IPFS, un file system decentralizzato il cui funzionamento è già stato descritto nel paragrafo precedente. Combinando l'uso di Ethereum e di IPFS è dunque possibile realizzare applicazioni totalmente decentralizzate la cui architettura risulta essere quella mostrata in figura.



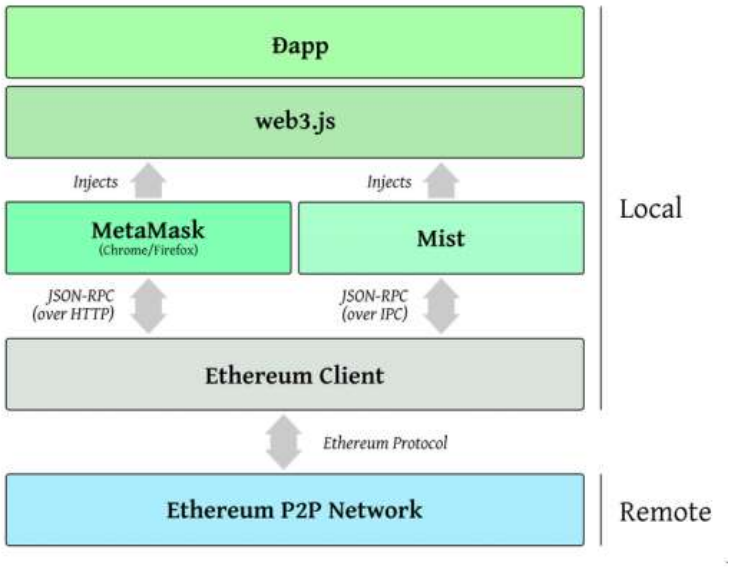
Tali misure precluderanno infatti l'utilizzo di un qualunque linguaggio di programmazione server-side tradizionale (come ad esempio PHP o ASP.NET) e di conseguenza l'applicazione web dovrà essere interamente sviluppata utilizzando solo linguaggi eseguibili client-side quali HTML e Javascript. Questa architettura prevede infatti che la computazione server-side dovrà essere totalmente realizzata dallo Smart Contract in esecuzione sulla blockchain.
Queste due tecnologie combinate possono quindi risultare sufficienti per realizzare gran parte delle applicazioni decentralizzate, con Ethereum che si occupa di memorizzare all'interno della blockchain lo stato corrente dell'applicazione, realizzando le stesse funzioni di un database, e di eseguirne la logica business, codificata all'interno di uno Smart Contract, grazie alla sua EVM (Ethereum Virtual Machine), mentre IPFS mantiene in modo persistente tutte le risorse statiche che non potrebbero essere memorizzate direttamente all'interno della blockchain in quanto risulterebbe estremamente costoso. Questo è dovuto al fatto che tutti i nodi della rete devono mantenere in locale una copia dell'intera blockchain, e se quest'ultima fosse utilizzata come un repository, invece che come un database per la memorizzazione di semplici transazioni, **la sua dimensione esploderebbe esponenzialmente nel giro di pochissimo tempo** divenendo completamente ingestibile. Pertanto all'interno della blockchain ci si limita a memorizzare dei riferimenti ai file, ovvero gli hash IPFS che identificano le risorse su di esso memorizzate.

La scelta di utilizzare una blockchain pubblica porta con sé lo svantaggio di richiedere il pagamento di una piccola commissione per l'esecuzione di ogni operazione, destinata a ricompensare i miner del lavoro computazionale svolto per aver validato una transazione ed aver eventualmente eseguito del codice di uno Smart Contract. Ciò può chiaramente risultare un problema per lo sviluppatore, fortunatamente su Ethereum è necessario pagare le commissioni di mining solo quando la computazione richiesta comporta un aggiornamento dei dati memorizzati all'interno della blockchain, mentre **se si opera in sola lettura tale pagamento non è richiesto**.

Questo comportamento è dovuto al fatto che un'operazione di aggiornamento deve essere eseguita necessariamente da tutti i nodi della rete, mentre per leggere i dati è sufficiente fare una richiesta al solo nodo a cui l'applicazione si connette direttamente e che funge come punto di ingresso alla rete Ethereum. **Limitare il numero di informazioni da immagazzinare e ridurli alle semplici informazioni di stato dell'applicazione,** costringe lo sviluppatore a trovare canali di comunicazione off-chain. Per questa ragione è stato introdotto Whisper, un protocollo di messaggistica peer-to-peer per applicazioni decentralizzate facente parte della suite di tecnologie sviluppate da Ethereum per la realizzazione del Web 3.0. Whisper, con il suo servizio di messaggistica decentralizzata, rappresenta l'ultimo elemento necessario per riuscire a sviluppare una Dapp arbitrariamente complessa, andando

quindi a completare il set di tecnologie necessarie per progettare l'architettura del sistema da realizzare.

Per poter utilizzare l'applicazione web decentralizzata l'utente dovrà disporre di un browser in grado di interagire con la blockchain di Ethereum. A tale scopo le soluzioni possibili sono molteplici, una delle quali è messa a disposizione direttamente dalla Ethereum Foundation, ovvero il browser Mist. **Mist** è un software che integra al suo interno tutti gli strumenti necessari per la gestione di un wallet, per il deployment di Smart Contract e persino un browser abilitato al web 3.0, ovvero all'utilizzo di Dapp, pensato appositamente per un pubblico di non addetto ai lavori. Mist è però un software ancora in fase di sviluppo, molto lento e con alcune criticità di sicurezza tuttora irrisolte, pertanto una possibile migliore alternativa, spesso suggerita, è rappresentata da **Metamask**, un'estensione per i browser tradizionali come Google Chrome, Mozilla Firefox e Opera che li abilita all'utilizzo delle Dapp, senza che l'utente abbia nemmeno la necessità di eseguire un client Ethereum sul proprio computer, come Geth o Parity. Metamask rende l'accesso al mondo delle dApps basate su Ethereum estremamente facile e alla portata di tutti. Anche Metamask mette a disposizione degli strumenti per una gestione completa dei propri wallet, consentendo di effettuare transazioni, firmare messaggi, aggiungere ed esportare chiavi private, gestire molteplici account e passare dalla rete principale di Ethereum a una qualsiasi testnet con un click. Sono molteplici dunque i layer che costituiscono l'architettura di un'applicazione web decentralizzata basata su Ethereum.



Grazie a **servizi come Infura**, è inoltre possibile evitare l'installazione e la configurazione di un client Ethereum e IPFS in locale, risparmiando così risorse sia in termini computazionali che di tempo e di denaro per la realizzazione dell'infrastruttura necessaria. Infura fornisce infatti agli sviluppatori di applicazioni decentralizzate strumenti ed API di facile utilizzo per consentire un accesso sicuro,

affidabile e scalabile a Ethereum e IPFS. Infura può essere utilizzato per la parte riguardante IPFS in quanto gli endpoints pubblici messi a disposizione da Infura per collegarsi alla rete Ethereum non espongono ancora le API necessarie per il funzionamento di Whisper. Per l'esecuzione di questa specifica applicazione è pertanto richiesto che l'utente abbia in esecuzione sul proprio computer un'istanza di un client Ethereum, come Geth o Parity, con il protocollo Whisper abilitato.

Configurazioni di Testnet

Per sviluppare applicazioni decentralizzate basate su Ethereum, è necessario per prima cosa configurare una blockchain di test che utilizzi una cripto valuta fittizia, ovvero un ambiente che consenta di **effettuare molteplici deployment di Smart Contract e transazioni di prova** senza che sia necessario sostenere un esborso economico reale. È possibile utilizzare sia testnet pubbliche online, che costruirsi delle proprie testnet private in locale. Quest'ultima possibilità è messa a disposizione direttamente **dal client Ethereum Geth ufficiale**.

Per configurare una nuova blockchain Ethereum in locale è necessario quindi creare un file JSON contente una serie di proprietà che caratterizzeranno la blockchain stessa, tra le quali si evidenziano:

- **difficulty**: valore scalare corrispondente al livello di difficoltà applicato durante il processo di mining di un blocco, e quindi proporzionale al tempo che sarà necessario per generarlo;
- **gaslimit**: valore scalare che definisce, a livello di blockchain, la quantità massima di gas utilizzabile per blocco;
- **chainId**: valore che identifica la catena corrente. Viene utilizzato per implementare una protezione contro i replay attack.

Un esempio di configurazione standard, è il seguente:

```
{
    "difficulty" : "0x80000",
    "extraData"  : "",
    "gasLimit"   : "0x8000000",
    "alloc": {},
    "config": {
        "chainId": 15,
        "homesteadBlock": 0,
        "eip155Block": 0,
        "eip158Block": 0
    }
}
```

Una volta creato il file di configurazione, è possibile procedere all'inizializzazione della blockchain, attraverso l'esecuzione del comando:

geth --datadir=./blockchain/ init ./genesis.json

Completato il processo di inizializzazione dovrà essere generato anche un primo account Ethereum che fungerà da coinbase, ovvero da wallet principale all'interno del quale Geth custodirà gli Ether falsi utilizzabili per i vari test, ottenuti come ricompensa per il mining. Dopo di chè sarà possibile lanciare l'esecuzione del client Ethereum specificando tutti i parametri che saranno necessari per il funzionamento dell'applicazione. Il comando da eseguire risulterà dunque essere il seguente:

geth --datadir=./blockchain/ --shh --rpc --rpcaddr "0.0.0.0" --rpcport "8545" --rpccorsdomain '*' --networkid=15 --rpcapi "db,eth,net,web3,shh,personal" --mine --minerthreads "1"

Di seguito viene descritto il significato di ciascun flag utilizzato:

- datadir: directory dove è memorizzata la blockchain;
- shh: abilita il protocollo Whisper;
- rpc: avvia l'interfaccia HTTP JSON-RPC;
- rpcaddr: interfaccia di ascolto del server HTTP-RPC;
- rpcport: porta di ascolto del server HTTP-RPC;
- rpccorsdomain: lista di domini da cui accettare richieste;
- networkid: identificatore della rete (deve corrispondere al chaidId affinché Metamask non sollevi eccezioni);
- rpcapi: elenco di API offerte attraverso l'interfaccia HTTP-RPC.
- mine: abilita il mining;
- minerthreads: numero di thread dedicati al processo di mining;

Con questa configurazione si esaurisce il lavoro necessario a creare l'infrastruttura tecnologica di test necessaria per il deployment dell'applicazione decentralizzata.

Occorre evidenziare i limiti che, ad oggi, le tecnologie utilizzate per la realizzazione di applicazioni decentralizzate come questa, ancora presentano.

Limiti dell'applicazione

Tra i limiti più importanti si evidenziano:

• **scalabilità limitata**: tra i più grandi vantaggi che si ottengono sviluppando un'applicazione decentralizzata vi è la mancata necessità di dover costruire, configurare e manutenere un'infrastruttura sulla quale rilasciare ed eseguire l'applicazione stessa, in quanto si sfrutta l'infrastruttura della rete preesistente scelta. Questa caratteristica può però risultare anche un enorme limite, infatti **il livello massimo di scalabilità dell'applicazione sarà determinato dalla capacità massima della blockchain scelta**.

In particolare, Ethereum in questo momento è in grado di gestire un throughput di **circa 15 transazioni al secondo**. È semplice verificare che una singola applicazione con volumi di dati sostenuti sarebbe in grado di saturare e congestionare l'intera rete in pochissimo tempo. **La blockchain di Ethereum non è quindi ancora pronta ad ospitare applicazioni reali con un consistente numero di utenti senza che non si verifichino gravi stalli della rete.** La scalabilità delle blockchain resta comunque uno dei temi più dibattuti e su cui si stanno concentrando la maggior parte degli sforzi della ricerca;

• **impossibilità di aggiornare facilmente gli Smart Contract**: dal momento che il codice sorgente degli Smart Contract è memorizzato all'interno della blockchain, questo diventa immutabile e permanente, rendendo di fatto impossibile aggiornare direttamente il contenuto di uno Smart Contract già pubblicato. Questo non impedisce in maniera assoluta l'aggiornamento di un'applicazione decentralizzata, ma lo complica notevolmente, costringendo gli sviluppatori a ricorrere a diversi stratagemmi più articolati che comportano la creazione di un nuovo contratto che dovrà interagire in qualche modo con il precedente, divenendo di fatto una specie di strato di secondo livello;

• **impossibilità di comunicare con l'esterno**: uno Smart Contract non può interagire con servizi offerti dal mondo esterno come email o notifiche sms. Se si desidera realizzare un'applicazione decentralizzata in grado di fornire anche questo tipo di funzionalità allora è necessario prevedere l'utilizzo di un server centrale che realizzi questi servizi in risposta ad eventi scatenati dallo Smart Contract per i quali rimane costantemente in ascolto. Inoltre, è da notare che uno Smart Contract può fornire risposte o scatenare eventi soltanto a seguito di richieste effettuate da un client, in quanto la sua esecuzione non può essere programmata per eseguire compiti periodici o avviare per primo una comunicazione (sono oggetti reattivi e non attivi);

• **impossibilità di memorizzare dati sensibili o confidenziali**: la scelta di utilizzare una blockchain pubblica come Ethereum consente da una parte di poter offrire all'utente una completa trasparenza sull'intero funzionamento dell'applicazione, dall'altra impedisce però di poter raccogliere dati sensibili o confidenziali in quanto memorizzandoli nella blockchain verrebbero esposti pubblicamente. Una possibile soluzione potrebbe essere quella di ricorrere alla crittografia, cifrando i dati prima di inviarli alla blockchain. Questo non impedirebbe però ad un eventuale attaccante di provare a violare la crittografia cercando di individuare la chiave utilizzata attraverso un attacco brute-force, che per quanto lento e dispendioso possa essere in base all'algoritmo scelto, può comunque non risultare un deterrente sufficiente in base al valore delle informazioni che l'attaccante sta cercando di carpire;

• **tempi di attesa**: la validazione delle transazioni, realizzata attraverso il processo di mining, richiede sempre un certo periodo di tempo, dipendente dal livello di congestione della rete in un dato momento e dal gas price configurato dall'utente. **Dal momento che la blockchain di Ethereum genera un nuovo blocco mediamente ogni 14 secondi**, l'utente dovrà attendere circa 10-20 secondi per vedere la propria transazione approvata (sia essa un trasferimento di Ether o l'invocazione di un metodo di uno Smart Contract che ne provoca un aggiornamento dello stato). Oggigiorno gli utenti hanno aspettative molto elevate quando navigano sul web e si aspettano che tutte le pagine e tutti i servizi vengano sempre erogati in modo praticamente istantaneo. L'attesa necessaria per il mining presente nelle applicazioni web decentralizzate può quindi compromettere parzialmente l'esperienza utente.

Risulta quindi chiaro che la blockchain non può essere impiegata in qualunque contesto e che presenta, ad oggi, ancora diversi problemi dovuti alla scarsa maturità della tecnologia.

Quando si realizza un'applicazione decentralizzata basata su blockchain, la fase di validazione ricopre un ruolo molto importante all'interno del ciclo di sviluppo, in quanto **una volta pubblicato sulla blockchain il codice sorgente di uno Smart Contract risulterà essere per sempre immutabile**. È quindi necessario accertarsi in maniera rigorosa della correttezza dei sorgenti, dal momento che eventuali bug saranno molto difficili da correggere.

Hyperledger Fabric

Tra le blockchain di ultima generazione (web 3.0), attualmente Ethereum detiene l'87% del mercato delle Dapp, a seguire troviamo la piattaforma EOS con il 19% e TRON con il rimanente 8%. Una analisi a parte occorre farla per le blockchain private in cui Hyperledger Fabric detiene il primato e le blockchain M2M (machine to machine) realizzate ad-hoc per applicazioni in cui sono richieste particolari requisiti prestazionali di cui ci occuperemo nell'ultima parte di questo documento.

Hyperledger è un framework realizzato dalla collaborazione tra la Linux Foundation e la comunità internazionale, atto a fornire un sistema completamente open-source per costruire soluzioni basate su blockchain che siano inseribili in modelli di business avanzati. Il principio di uso di Hyperledger è analogo a quello di un qualsiasi framework open-source, quindi all'utente che voglia interfacciarvisi è fornita una documentazione base e una serie di componenti installabili sul proprio PC ed estensibili a piacere, con i quali realizzare la propria rete. Hyperledger Fabric è una delle varie implementazioni possibili di Hyperledger e permette la realizzazione di blockchain permissioned dove i nodi vengono gestiti dal proprietario della rete, tipicamente un'azienda singola o un gruppo di aziende, il quale è l'unico con l'autorità per permettere a questi di instaurare una comunicazione tra loro e partecipare al network. Le parti che contribuiscono alle transazioni non sono solo singoli utenti ma anche organizzazioni o aziende, le quali possono possedere uno o più nodi; quello che si realizza così è un consorzio dove le aziende hanno la possibilità di gestire e configurare i nodi ad esse forniti. I nodi possono poi anche essere configurati per permettere ad utenti esterni generici di visualizzare le informazioni o parte di esse, tramite un livello di accesso personalizzato, in modo che queste possano essere comunicate anche all'esterno del network in maniera sicura; in questo caso parliamo di end nodes. La particolarità di Fabric sta nel permettere la creazione di canali distinti all'interno dello stesso nodo, fornendo quindi la possibilità effettiva di garantire differenti livelli di accesso e permessi ad utenti differenti riguardo a informazioni diverse; i canali così creati sono realizzabili per essere del tutto separati anche se presenti sullo stesso nodo, come se stessimo effettivamente realizzando due catene di blocchi parallele che vengono estese in maniera del tutto indipendente l'una dall'altra. E inoltre possibile, esattamente come in Ethereum, definire degli Smart Contract che stabiliscano regole di funzionamento del sistema.

La facilità offerta dai canali, unita a quella degli Smart Contract, garantisce la possibilità di abilitare una grande varietà di modelli di business tramite Fabric; test interni di IBM hanno dimostrato come sia possibile mantenere contemporaneamente fino a 25 canali sullo stesso nodo. Nell'implementazione di Fabric tramite canali ogni utente possiede un certificato con il quale si presenta ai nodi al momento della richiesta di effettuare una transazione; all'interno del certificato

sono elencati i suoi permessi e questo viene confrontato con lo Smart Contract, in questo caso detto Chaincode, per garantire che l'operazione richiesta con la transazione sia permessa all'utente in questione.

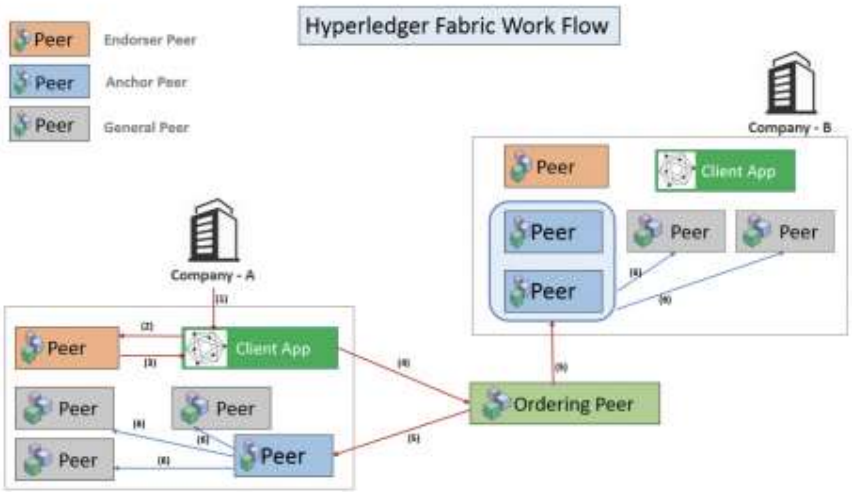
In Fabric i nodi possono essere di tre tipi:

1. Endorser: hanno il compito di ricevere le richieste di transazioni dalle applicazioni fornite ai vari client. Una volta ricevuta la richiesta validano la transazione (controllando i certificati allegati), eseguono il Chaincode e simulano il risultato della transazione, senza però modificare il ledger, per capire se questa debba essere approvata o rifiutata;

2. Anchor: sono i nodi responsabili di ricevere gli aggiornamenti e comunicarli agli altri nodi facenti parte dello stesso canale. Questo tipo di nodi può essere "scoperto" da parte degli altri nodi senza l'intervento diretto dell'organizzazione che ha configurato la rete;

3. Orderer: sono i nodi che si occupano della comunicazione all'interno del network. Questi mantengono e modificano lo stato del ledger, creando i blocchi e inviandoli in broadcast a tutti i peer. L'implementazione più comune per questo avviene tramite Kafka, un software di scambio messaggi con alta "throughput fault tolerance".

Il flusso di processo di una generica transazione su Fabric è il seguente: un partecipante di una organizzazione invia una richiesta di transazione al proprio nodo endorser, il quale esegue il Chaincode e simula la transazione, rispondendo al client con il responso "transazione accettata" o "transazione rifiutata"; a questo punto se la transazione è stata accettata il client la invia ai nodi orderer che la includono in un blocco e la inviano in broadcast a tutti i nodi anchor delle differenti organizzazioni della rete, i quali poi la ritrasmetteranno ai vari peer interni per sincronizzare i ledger.



Per raggiungere il consenso tra i vari nodi orderer viene usato un algoritmo elettivo chiamato Raft, il quale si basa su una strategia ad elezione, dove in fase di configurazione i nodi convergono alla decisione di un "master" che riceverà le future transazioni e le ritrasmetterà in broadcast a tutti gli altri.

L'implementazione di Fabric permette di conservare due tipologie di dati:

- On-chain;
- Off-chain.

Mentre i primi sono dati riguardanti le transazioni e sono salvati in blocchi con un identificativo unico e una hash, i secondi sono documenti di qualsiasi tipo, foto, video, file pdf o in formato testuale; questa caratteristica permette di usare Fabric per scambiare e conservare come in un vero e proprio databse anche i documenti che devono essere scambiati tra utenti appartenenti a diverse organizzazioni all'interno dello stesso consorzio.

Hyperledger, nella sua implementazione Fabric, non consente la possibilità di effettuare delle fork sulla catena, obbligando quindi tutti i nodi a mantenere sempre la stessa copia delle informazioni e garantendone quindi la completa consistenza. E altresì possibile utilizzare Fabric per realizzare blockchain pubbliche, creando un singolo certificato per un utente generico, che possa essere riutilizzato da tutti per proporre transazioni alla rete e che passi la verifica del Chaincode; vista la presenza dei canali è anche possibile far coesistere sullo stesso nodo una blockchain privata e una pubblica.

Al contrario di Bitcoin, Fabric espone una serie di funzionalità che lo rendono compatibile con un utilizzo a livello industriale. La caratteristica di versatilità di Hyperledger e la sua varietà di implementazioni sono un grande incentivo al suo sviluppo e alla sua adozione poiché permettono un livello di personalizzazione avanzato e la possibilità di adattare la rete alle più disparate esigenze. Inoltre, il fatto di poter realizzare canali privati insieme a canali pubblici, rende possibile soddisfare la richiesta, da parte delle aziende, di riservatezza riguardo a dati sensibili ad uso prettamente interno; contemporaneamente è però possibile soddisfare, almeno in parte, la richiesta del pubblico di visualizzare le informazioni in maniera trasparente, eliminando il paradigma classico per il quale è necessario fidarsi di chi fornisce le informazioni, il quale tuttavia molto spesso è direttamente interessato a manipolarle od ometterne una parte.

La maggior parte delle piattaforme blockchain che supportano Smart Contract seguono un'architettura **order-execute** nella quale il protocollo di consenso valida e ordina le transazioni, poi le propaga a tutti i nodi, infine i peer eseguono le transazioni sequenzialmente. L'architettura order-

execute può essere ritrovata in quasi tutti i sistemi blockchain esistenti, dalle piattaforme pubbliche/permissionless quali Ethereum a piattaforme private/permissioned come Corda. Fabric introduce una nuova architettura per le transazioni chiamata **execute-ordervalidate**, la quale si prefigge come obiettivi: resilienza, flessibilità, scalabilità, prestazioni e confidenzialità.

Il flusso delle transazioni è diviso in tre passi:

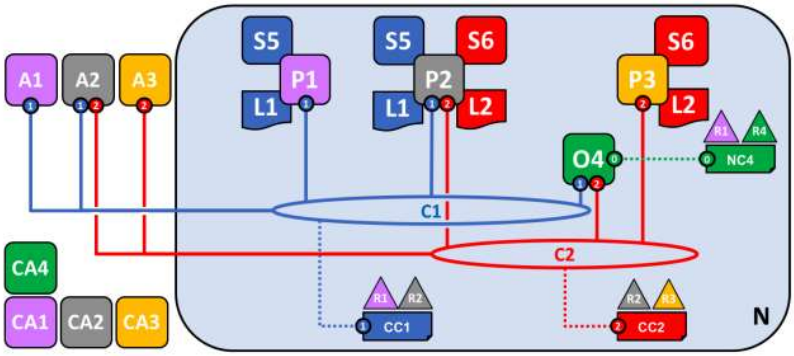
• Esecuzione di una transazione e controllo della sua correttezza.

• Ordinamento delle transazioni tramite un protocollo di consenso.

• Validazione delle transazioni, in accordo a una specifica politica di approvazione, prima di farne il commit nel ledger.

In Fabric, ogni transazione necessita di essere eseguita solo dal sottoinsieme di nodi necessari per soddisfare la propria specifica politica di approvazione. Questo permette l'esecuzione di transazioni in parallelo, migliorando le prestazioni e la scalabilità del sistema. La prima fase elimina anche il non determinismo, in quanto risultati inconsistenti possono essere filtrati prima di essere ordinati.

Ruoli chiave sono svolti dall'orderer e dai canali: tramite i canali si può accedere ai chaincode, i quali permettono di effettuare operazioni sul ledger associato a ogni canale; ogni transazione viene approvata dai peer che ospitano i chaincode e poi inviata all'orderer per essere validata.



In figura, le organizzazioni R1, R2, R3 e R4 hanno deciso congiuntamente di mettere per iscritto un accordo per organizzare e utilizzare una rete N sviluppata con Hyperledger Fabric. R4 è l'iniziatore della rete (cioè colui che ha costruito la versione iniziale della rete) e non ha intenzione di effettuare transazioni di business su di essa; non avendo dunque "interessi" sulla rete, è l'organizzazione perfetta per gestire l'orderer. Ognuna delle organizzazioni ha una Certificate Authority preferita (CA1, CA2, CA3, CA4). R1 e R2 hanno bisogno di comunicazioni private all'interno della rete, come pure R2 e R3, le quali sono rese possibili grazie alla creazione dei canali C1 e C2. L'organizzazione R2 ha un'applicazione client che può lavorare su C1 e C2, mentre l'organizzazione R3 ha un'applicazione

che invece lavora solo su C2. Il nodo peer P1 mantiene una copia del ledger L1 associato a C1 e ospita il chaincode S5 per le invocazioni su C1. Il nodo peer P2 mantiene una copia del ledger L1 e una copia del ledger L2 associata a C2, oltre ai chaincode S5 e S6 per C1 e C2. Il nodo peer P3 mantiene una copia del ledger L2 e ospita il chaincode S6. La rete è governata in accordo a policy specificate nella configurazione di rete NC4, la quale specifica che il controllo sulla rete è nelle mani delle organizzazioni R1 e R4. Il canale C1 è governato in accordo alle policy specificate nella configurazione di canale CC1 ed è sotto il controllo delle organizzazioni R1 e R2. Il canale C2 è governato in accordo alle policy specificate nella configurazione di canale CC2 ed è sotto il controllo delle organizzazioni R2 e R3. Il servizio di ordinamento O4 funziona da punto di amministrazione per N e utilizza il canale di sistema per mantenere le impostazioni di rete. Il servizio di ordinamento supporta anche i canali applicativi C1 e C2 al fine di ordinare le transazioni in blocchi da distribuire ai peer. Sul sito di Hyperledger è possibile accedere a molti progetti sample per testare reti già definite e pronte per la sperimentazione.

Progetto Tenderfone (UNIBO)

In questo paragrafo, vengono analizzati opportunità ed ostacoli nell'estendere o integrare i benefici di due sistemi concepiti per risolvere compiti specifici: sistemi agent-oriented e le blockchain.

Il ruolo centrale nell'unione di questi due sistemi lo rivestono gli Smart Contract. Come già accennato nei paragrafi precedenti gli SC consentono alla blockchain di funzionare come motore di calcolo distribuito e generico ma allo stesso tempo conservando le proprietà principali di una BC in materia di sicurezza, fiducia e tolleranza ai guasti. Gli SC in realtà estendono la portata delle BC per assolvere funzioni di calcolo oltre l'archiviazione e la gestione dei dati. Allo stato attuale, i contratti intelligenti vengono applicati principalmente in tre casi d'uso: automazione delle procedure, verifica delle proprietà di transazioni e interazioni di mediazione tra parti distribuite non attendibili.

I contratti intelligenti vengono sfruttati per eseguire calcoli "algoritmici" puri o calcoli interattivi volti a governare quando, come e in che misura le entità off-chain possono comunicare. Si può intuire quindi come l'interazione tra MAS e BC sia un campo di ricerca promettente, aumentano l'espressività e l'efficacia del calcolo di sistemi concorrenti.

Se da un lato, gli agenti sono entità autonome, distribuite, le cui interazioni dovrebbero essere governate e mediate in modo affidabile; dall'altro, le blockchain e contratti intelligenti sono strumenti sensibili alla fiducia per mediare e governare l'interazione tra diverse entità.

Prima dell'avvento dei contratti intelligenti, i calcoli in una blockchain erano assegnati ad entità di calcolo off-chain, come processi software dedicati che funzionavano come "client" blockchain, mentre con le Smart Contract tali processi possono essere effettivamente trasferiti all'interno degli stessi nodi blockchain, pertanto gli agenti potrebbero essere inseriti "nella blockchain", magari come nuovo modello computazionale per Smart Contract.

La maggior parte degli approcci utilizzati fino ad ora affiancavano i MAS alle blockchain, limitandoli ad interazioni opportunistiche, ovvero si limitavano alle richieste di servizio del primo al secondo. Cosa succede invece se sostituiamo un SC con un agente? Lo studio compiuto dall'Università di Bologna cerca di indagare questa opportunità.

Si consideri ad esempio il caso di un'azienda manifatturiera disposta a vendere i propri prodotti all'estero. Per fare ciò, occorre mettere in atto un'articolata catena di approvvigionamento che coinvolge, ad esempio, una compagnia di spedizioni, una società di distribuzione e un negozio al dettaglio. In tale scenario, l'azienda produttrice potrebbe sfruttare i contratti intelligenti per

(i) monitorare i movimenti dei suoi prodotti lungo la catena di approvvigionamento e

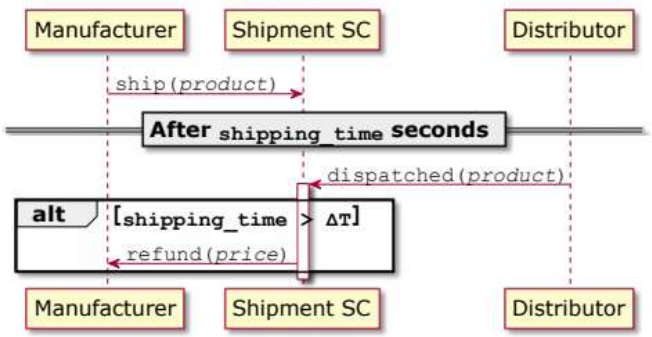
(ii) ricevere un rimborso automatico nel caso in cui alcune delle sue società partner non riescano a spedire i prodotti puntualmente.

In particolare, viene istanziato un contratto intelligente per ogni fase della catena di approvvigionamento.

Se ci concentriamo sul caso della compagnia di spedizioni, uno smart contract può essere programmato per

- mantenere traccia dello stato di spedizione dei prodotti e
- rimborsare automaticamente l'azienda produttrice se i prodotti pervengono alla società di distribuzione dopo un periodo di tempo concordato dalla spedizione.

Questo è esattamente ciò che è rappresentato nella Figura



dove l'interazione di una spedizione SC con una entità off-chain è descritta tramite UML. In sostanza, i diagrammi rappresentano un possibile progetto di questo scenario in aggiunta a un contratto intelligente Ethereum o HLF. Al momento della spedizione, un'entità fuori catena che rappresenta il Fabbricante invoca un metodo nave(...) sulla SC di Spedizione, notificando la consegna del prodotto.

Di conseguenza, l'SC registra l'ora di inizio della spedizione. Ogni volta che il prodotto raggiunge effettivamente il Distributore, quest'ultimo, che è ancora un'entità fuori catena, certifica il successo della spedizione del prodotto invocando il metodo spedito(...) sulla Spedizione SC. Reagendo a tale invocazione del metodo, la SC decide se un rimborso deve essere eseguito o meno, a seconda che il tempo totale necessario per la spedizione è maggiore o meno di una determinata soglia (ΔT).

Questo scenario evidenzia, ad esempio, i limiti espressivi del mainstream SC verso la proattività e, quindi, dell'autonomia, tratto distintivo dell'agentività. Se la Spedizione SC fosse dotata di autonomia di calcolo, sarebbe stata in grado di effettuare un rimborso automatico non appena la durata della spedizione avrebbe superato la soglia ΔT, senza richiedere alcun intervento esterno.

Al contrario, data l'attuale natura degli SC mainstream, che sono intrinsecamente reattivi ad entità off-chain, la Shipment SC può eseguire il rimborso solo dopo che una terza parte, vale a dire il Distributore, lo innesca. Questo in pratica è un problema poiché **il mainstream SC non ha la capacità effettiva di far rispettare le regole** per cui sono state istanziate, se nessuna entità esterna le attiva.

Questa problematica è stata affrontata e risolta in letteratura attraverso la definizione degli "**oracoli**". Gli oracoli sono essenzialmente, parti fidate di tipo off-chain incaricate di fornire input freschi e tempestivi ai contratti intelligenti affinché possano realizzare la loro logica di business.

L'approccio basato sull'oracolo naturalmente è solo una soluzione. In effetti, le suddette limitazioni rivelano che i contratti intelligenti hanno bisogno di uno spostamento paradigmatico verso un più alto grado di autonomia computazionale. L' orientamento verso l'agente può essere una risposta a questa esigenza, poiché il paradigma agent-oriented abbraccia esplicitamente l'autonomia di entità computazionali.

La maggior parte degli approcci che tentano di collegare MAS e tecnologie blockchain (BCT) sono del tipo "**agent-vs-blockchain**", ovvero mettono fianco a fianco un MAS e una blockchain, lasciando che gli agenti sfruttino i servizi blockchain al bisogno, in modo opportunistico. In questo approccio semplicemente il MAS sfrutta il blockchain come utilizzerebbe qualsiasi altra libreria software. In effetti, gli agenti sono essenzialmente trattati come qualsiasi altra entità off-chain, ovvero come i

client blockchain, la cui interazione con la blockchain è limitata all'emissione di transazioni e all'implementazione/invocazione di contratti intelligenti, come farebbero con qualsiasi altro servizio.

Una famiglia di approcci più interessante è quella che chiamiamo "**agent-to-blockchain**", ovvero sforzi che cercando di inserire elementi di modelli e tecnologie orientati agli agenti direttamente nella blockchain.

Ad esempio, l'adozione di linguaggi di programmazione ad agenti per l'implementazione di contratti intelligenti sarebbe un sforzo di integrazione appartenente a questa categoria, oltre al potenziamento computazionale dei modelli di contratti intelligenti (di solito, orientato agli oggetti) con caratteristiche che definiscono l'orientamento dell'agente (ad esempio, proattività).

Quindi, è possibile un'altra famiglia di approcci, "**blockchain-to-agent**", che includerebbe approcci di integrazione che cercano di iniettare negli agenti concetti e tecniche appartenenti al regno delle BCT.

In quest'ottica nasce il progetto Tenderfone che realizza una blockchain privata di tipo permissioned la quale supporta l'implementazione e l'esecuzione di **Smart Contract proattivi e logici**.

Il sistema è concettualmente composto da:

- un insieme di **nodi**, connessi in modo peer-to-peer, che eseguono collettivamente un protocollo di consenso per comportarsi come un'unica unità di elaborazione. Offrono anche un ambiente per l'implementazione e l'esecuzione di contratti intelligenti.
- un'autorità di **certificazione (CA)**, che ha il compito di rilasciare certificati sia agli utenti finali che ai nodi. Conserva la natura centralizzata del sistema di accreditamento, necessaria per identificare in modo univoco e sicuro ciascuna entità e per rilevare le transazioni contraffatte.
- un **client**, inteso a mediare l'interazione tra utenti finali e nodi. Più precisamente, consente agli utenti finali di creare e trasmettere transazioni al sistema volte a creare, invocare o eliminare contratti intelligenti.

A causa del loro duplice compito, ogni nodo è progettato e implementato come due processi diversi:

- un processo *centrale* , responsabile di ciò che riguarda il consenso con gli altri nodi e la gestione della blockchain;
- un processo di *interprete logico* , responsabile della gestione degli Smart Contract durante tutto il loro ciclo di vita.

Tenderfone si affida a Tendermint per quanto riguarda il core process di ogni nodo di sistema. Il LogicInterpreter è scritto in Java; si basa su JTendermint per interagire con il core Tendermint e utilizza tuProlog come motore logico per gli Smart Contract. La CA è scritta in Javascript (Nodo) mentre il Cliente è scritto in Typescript.

L'implementazione del sistema **è ancora in una fase prototipale ed è il primo prototipo di blockchain orientata agli agenti**.

Questa nuova implementazione di blockchain, anche se è ancora in fase prototipale, è al centro di diversi lavori di ricerca per le sue caratteristiche. Infatti possiede una maggiore espressività degli Smart Contract, potenziati dall'autonomia, attraverso meccanismi che li dotano di un flusso di controllo e da una comunicazione asincrona (attraverso code di messaggi), ovvero si propongono di realizzare contratti intelligenti utilizzando l'astrazione dell'agente.

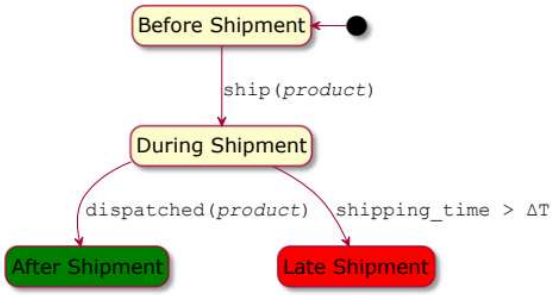
Tenderfone è dotato di un linguaggio dichiarativo basato sulla logica del primo ordine (FOL). In particolare, adotta la sintassi Prolog; quindi, il suo interprete è un motore Prolog: tuProlog.

Si sfrutta la programmazione logica come mezzo per dotare il contratto intelligente di ragionamenti orientati agli obiettivi. Data questa scelta, un Tenderfone SC è in realtà una teoria la cui logica di contro per obiettivi, espressi come termini Prolog, può essere dimostrata. O, in altri termini, è una base di conoscenza che descrive lo stato dell'applicazione e consente di verificare le proprietà di tali stati (es. verificare che le transazioni rispettino alcuni vincoli).

Come discusso nel paragrafo precedente, lo scenario della catena di approvvigionamento è di fondamentale importanza nell'ambito dell'IoT basato su BCT. Nello scenario descritto, l'entità off-chain incaricata di riconoscere la spedizione dei prodotti, ovvero l'azienda distributrice, rappresenta un unico punto di fiducia della blockchain. Di fatto, nel peggiore dei casi può ritardare arbitrariamente il suo riconoscimento, impedendo così il rimborso alla società produttrice.

Sfruttando le funzionalità legate al tempo di Tenderfone per rendere autonomo il contratto intelligente si riesce a rimborsare l'azienda produttrice non appena la durata della spedizione supera una determinata soglia. Ciò elimina la necessità di definire un'entità off-chain incaricata di attivare il rimborso in caso di ritardo spedizione o l'utilizzo di un oracolo.

In altre parole, **l'SC Tenderfone può essere modellato come l'automa a stati finiti temporizzato rappresentato in Figura**



dove il passaggio allo stato di "Late shipping" può essere provocato dal trascorrere del tempo, automaticamente. Il listato dello SC che riproduce questo comportamento è riportato di seguito:

```
init(Duration) :-
  set_data(state, before_shipment),
  set_data(expected_duration, Duration).

receive(ship(product), manufacturer) :- self(Me),
  get_data(state, before_shipment),
  get_data(expected_duration, DT),
  set_data(due(manufacturer, Me), euro(13))
  set_data(state, during_shipment),
  delay(DT, late).

receive(late, Me) :- self(Me),
  get_data(state, during_shipment),
  set_data(state, late_shipment),
  set_data(due(manufacturer, Me), 0),
  set_data(due(Me, manufacturer), euro(13)),
  send(refund(euro(13)), manufacturer).

receive(dispatched(product), distributor) :-
  get_data(state, during_shipment),
  set_data(state, after_shipment).
```

Il funzionamento del SC nel listato è abbastanza semplice. Si assume di conoscere le identità sia del produttore che del distributore, oltre all'ID del prodotto che deve essere spedito. La durata prevista della spedizione viene fornita al SC al momento dell'implementazione.

Successivamente, la SC

- attende che l'azienda produttrice avvii la procedura di spedizione;
- non appena avviata la procedura di spedizione, viene schedulato il rimborso automatico
- quindi, si attende che l'azienda distributrice confermi l'invio del prodotto.

L'esito finale della procedura di spedizione dipende da quale evento si verifica per primo: nel caso in cui avviene prima il riconoscimento del distributore, non viene effettuato alcun rimborso; in caso contrario, il rimborso è automatico, mentre il riconoscimento tardivo viene ignorato.

Progetto Vimana (UNIME)

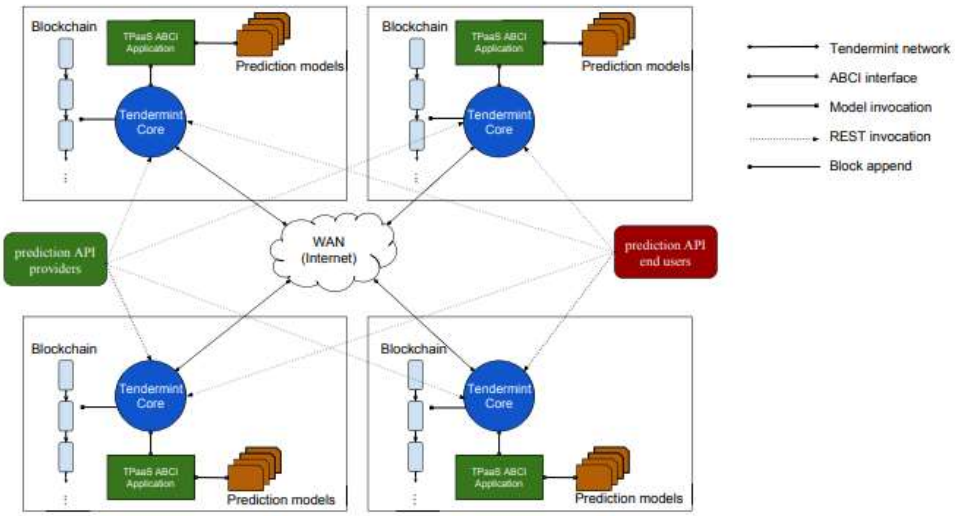
Diversamente dal problema affrontato nel progetto Tenderfone, dove l'obiettivo era introdurre proattività e flessibilità negli Smart Contract, in questo lavoro il focus del progetto è realizzare un nuovo modello di business basato su tecnologia blockchain attraverso l'implementazione di un **framework open source Prediction-as-a-Service decentralizzato**. Abbiamo già accennato nei paragrafi precedenti al **Sensing-as-a-Service** come modello di business correlato a particolari tipi di Blockchain M2M in cui un dispositivi IoT sono elementi attivi della rete quali fornitori di servizi e svolgono la loro funzione in cambio di una remunerazione. Il progetto Vimana introduce il nuovo paradigma **Prediction-as-a-Service**, che porta i vantaggi del modello di business **Software-as-a-Service** nel mondo delle API di previsione. Il modello di business Software-as-a-Service è proprio quello descritto dal ruolo dei miners all'interno delle blockchain pubbliche. I miners ottengono ricompense per risolvere un difficile problema computazionale basato su un algoritmo di hashing; la soluzione al problema, chiamato proof-of-work, è inclusa nel nuovo blocco ed agisce come prova che il miner ha impiegato uno sforzo computazionale per arrivare alla soluzione. Vincendo questa gara, si ottiene la ricompensa ed il diritto di registrare le transazioni nella blockchain. Questa stessa logica quindi può essere applicata a funzionalità di più alto livello, ovvero alle Dapp.

Prediction-as-a-Service è un paradigma emergente in cui i modelli addestrati (machine learning) sono ospitati in un Cloud data center e forniti tramite API per effettuare previsioni sui dati raccolti (es., Google Cloud Machine Learning Engine).

Il progetto Vimana nasce con l'obiettivo di realizzare una infrastruttura che consenta di realizzare un mercato che coinvolge i seguenti attori: (i) fornitori di API di previsione; (ii) fornitori di servizi cloud; (iii) utenti finali dell'API di previsione. I fornitori di API di previsione sono sviluppatori di machine learning che addestrano i propri modelli di previsione e desiderano monetizzarli. Analogamente a quanto accade in uno scenario Software-as-a-Service, possono sfruttare le infrastrutture dei provider Cloud per ospitare i loro modelli e offrirli come servizio agli utenti finali. A seconda del modello di business, i provider cloud possono trasferire valuta ai fornitori di API di previsione o agli utenti finali dell'API di previsione, in base alle richieste. Nello scenario sopra descritto, si possono identificare diversi possibili attacchi/comportamenti scorretti che l'infrastruttura Vimana cerca di risolvere. In realtà l'infrastruttura realizzata, oltre ai meccanismi sopracitati, offre l'opportunità di spostare l'elaborazione dati dai Cloud verso i nodi edge, sfruttando i vantaggi dell'infrastruttura blockchain e del protocollo di consenso per garantire l'immunità contro specifici comportamenti scorretti.

La figura sottostante mostra l'architettura del framework proposto nel caso in cui quattro Cloud provider aderiscano al consorzio TPaaS. È stata sviluppata un'applicazione Tendermint che implementa una specifica interfaccia ABCI che gestisce due diverse tipologie di utenti e le loro transazioni.

I fornitori di API di previsione possono interagire con il sistema tramite una specifica API REST caricando nuovi modelli di previsione e rendendoli disponibili al pubblico. In questo caso specifico, l'applicazione ABCI verifica se l'utente ha permesso di caricare un modello e di archiviarlo in memoria pronto a ricevere richieste di inferenza. Questi ultimi vengono inviati dagli utenti finali dell'API di previsione tramite un'API REST. In questo modo, gli utenti attivano l'invocazione di un modello di previsione inviando dati di input appropriati e ricevono i risultati di tali invocazioni quando l'inferenza è stata eseguita. Entrambi i tipi di utenti possono interagire con uno qualsiasi dei nodi del consorzio. Ogni volta che viene caricato un nuovo modello o viene ricevuta e soddisfatta una nuova richiesta di inferenza, lo stato dell'applicazione viene aggiornato, viene generato un nuovo hash dello stato che viene inserito in un nuovo blocco. Anche in questo caso si ricorre al framework Tendermint Core che è responsabile di: (i) inviare opportuni messaggi all'interfaccia TPaaS ABCI per ogni transazione ricevuta (richiesta da un utente esterno) affinché possano essere elaborati secondo il comportamento sopra descritto; (ii) diffondere le transazioni ad altri nodi in modo che il modello di inferenza sia invocato ed eseguito su tutti i nodi; (iii) consentire ai nodi di raggiungere il consenso su tutti i blocchi blockchain. In questo modo, la blockchain memorizza la cronologia passata dell'applicazione dal suo avvio fino al momento presente e il consenso viene raggiunto ad ogni transizione di stato in modo che lo stato dell'applicazione sia correttamente replicato su tutti i nodi.



Tendermint

Tendermint fornisce una infrastruttura software che consente agli sviluppatori di realizzare nuove soluzioni blockchain. Fondamentalmente Tendermint è un motore di consenso ad alte prestazioni e scalabile, dove la logica dell'applicazione è tenuta separata dalla logica del consenso. La separazione netta tra questi tue elementi del sistema, consente di iniettare una logica personalizzata nelle applicazioni blockchain, **andando ben oltre il concetto tradizionale degli Smart Contract**. Esse possono usare **qualsiasi tipo di software aziendale** per gestire scenari applicativi complessi, ed ha dato il via ad innumerevoli progetti, come quelli descritti in precedenza, consentendo lo sviluppo delle application-specific blockchain.

Le **application-specific blockchain** sono blockchain personalizzate realizzate ad oh per far funzionare una singola applicazione, invece di creare un'applicazione decentralizzata su una blockchain sottostante comune come ad esempio Ethereum basata su macchine virtuali in grado di interpretare programmi completi chiamati Smart Contract. Questi **Smart Contract sono molto utili per casi d'uso come eventi una tantum** (ad es. ICO), ma possono non essere all'altezza della creazione di piattaforme decentralizzate complesse.

Gli Smart Contract sono generalmente sviluppati con linguaggi di programmazione specifici che possono essere interpretati dalla macchina virtuale sottostante. Questi linguaggi di programmazione sono spesso immaturi e intrinsecamente limitati dai vincoli della macchina virtuale stessa. Ad esempio**, la macchina virtuale Ethereum non consente agli sviluppatori di implementare l'esecuzione automatica del codice**. Gli sviluppatori sono anche limitati al sistema basato su account dell'EVM e possono scegliere solo da un insieme limitato di funzioni per le loro operazioni crittografiche. Questi sono esempi, ma suggeriscono la mancanza di flessibilità che spesso comporta un ambiente basato su Smart Contract. Gli Smart Contract sono tutti gestiti dalla stessa macchina virtuale, ciò significa che competono per le stesse risorse, il che può limitare gravemente le prestazioni. Anche se la macchina a stati dovesse essere suddivisa in più sottoinsiemi, gli Smart Contract dovrebbero comunque essere interpretati da una macchina virtuale, il che limiterebbe le prestazioni rispetto a un'applicazione nativa implementata a livello di macchina a stati. Un altro problema, derivante dalla condivisione dello stesso ambiente sottostante, è la conseguente limitazione sul controllo. Un'applicazione decentralizzata è un ecosistema che coinvolge più attori. Se l'applicazione è costruita su una blockchain di macchine virtuali di uso generale, le parti interessate

hanno un controllo molto limitato sulla loro applicazione, fortemente influenzata dalla governance della blockchain sottostante. Se c'è un bug nell'applicazione, si può fare ben poco.

Una blockchain basata su Tendermint permette la replica coerente e sicura e di un'applicazione su macchine diverse. Sicura perché l'applicazione continua a funzionare anche se 1/3 delle macchine si guasta arbitrariamente (capacità nota come tolleranza d'errore bizantina-BTF) e coerente perché ogni macchina vede le stesse transazioni e si trovano nello stesso stato.

L'algoritmo per il consenso ad alte prestazioni su cui è realizzata l'infrastruttura è noto come PBFT, ovvero Practical Byzantine Fault Tolerance.

L'algoritmo di consenso BFT è uno dei più vecchi algoritmi di consenso. Apparso per la prima volta nel 1999, prende il nome dal Problema dei Generali Bizantini in cui alcuni Generali non sapevano se attaccare o meno a causa di informazioni discordanti ricevute dal comandante e dagli altri Generali. Il problema sottendente che alcuni Generali erano traditori e quindi le informazioni che trasmettevano erano falsate. La soluzione del problema è stata affidata ad un alto numero di messaggi tra i partecipanti. L'ordine corretto è quello alla cui versione aderisce la maggioranza.

Applicato alle reti distribuite come la blockchain il Practical Byzantine Fault Tolerance consocia i nodi in reti minori. All'interno di ciascun gruppo, la maggioranza dei nodi vota il nodo leader che sulla base delle informazioni ottenute dai nodi consociati verifica i blocchi. L'obiettivo del sistema è di impedire a nodi malevoli di partecipare alla creazione dei blocchi, rendendo la blockchain meno esposta ad attacchi.

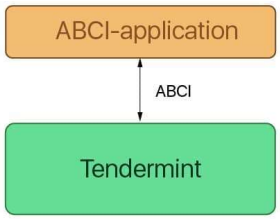
Tendermint offre prestazioni eccezionali, il consenso di Tendermint può elaborare migliaia di transazioni al secondo, con latenze di commit dell'ordine di uno o due secondi. In particolare, le prestazioni di oltre un migliaio di transazioni al secondo vengono mantenute anche in condizioni conflittuali difficili, con i validatori che si bloccano o trasmettono voti fraudolenti.

I principali vantaggi sono:

- Può gestire il volume delle transazioni alla velocità di 10.000 transazioni al secondo per transazioni fino a 250 byte.
- Sicurezza migliore e più semplice per il client che diventa più leggero rendendolo ideale per dispositivi mobili e IoT. Al contrario, i client leggeri Bitcoin richiedono molto più lavoro e hanno molte richieste che lo rendono poco pratico per determinati casi d'uso.
- Tendermint utilizza una fork-accountability che blocca gli attacchi come long-range-nothing-at-stake double spends e gli censorship.

- Tendermint è implementato tramite un "motore di consenso indipendente dall'applicazione". Fondamentalmente può trasformare qualsiasi applicazione black box deterministica in una blockchain replicata in modo distribuito.

Tendermint è divisibile in due parti principali: Tendermint Core, ossia la parte che gestisce il "motore" della blockchain, e l'Application Blockchain Interface (ABCI), che permette alle transazioni di essere gestite da una logica applicativa scritta in qualunque linguaggio di programmazione. La parte core di Tendermint garantisce, invece, che le transazioni vengano memorizzate su ogni macchina nello stesso ordine.



Progetti diversi hanno esigenze diverse. Alcuni progetti devono avere un sistema aperto in cui chiunque può partecipare e contribuire, come Ethereum. D'altra parte, abbiamo organizzazioni come l'industria medica, che non possono esporre i propri dati a tutti, allora come può Tendermint aiutare a soddisfare entrambe queste esigenze?

Tendermint rappresenta soltanto il primo strato dell'applicazione, cioè quella delegata a gestire solo il networking ed il meccanismo di consenso per la blockchain.

Quindi, si occupa di:

- Propagazione della transazione tra i nodi tramite il protocollo gossip
- Aiuta i validatori a concordare l'insieme di transazioni che vengono aggiunte alla blockchain.

Ciò significa che il livello dell'applicazione il programmatore è libero di definire come viene gestito il set di validatori all'interno dell'ecosistema. Gli sviluppatori possono consentire all'applicazione di avere un sistema elettorale che elegge i validatori in base a ai loro token nativi creando quella che viene definita **Proof-of-stake** per una blockchain pubblica. Inoltre, gli sviluppatori possono creare un'applicazione che definisce un insieme ristretto di validatori pre-approvati che si occupano del consenso per i nuovi nodi che entrano nell'ecosistema. Questa è chiamato **proof-of-authority** ed è il meccanismo di consenso distintivo di una blockchain autorizzata o privata.

L'implementazione del proof-of-stake di Tendermint è molto più scalabile di un tradizionale algoritmo di consenso proof-of-work. Il motivo principale è che i sistemi basati su POW non possono

eseguire lo sharding. Lo sharding fondamentalmente partiziona orizzontalmente un database e crea database o frammenti più piccoli che vengono quindi eseguiti in parallelo dai nodi.

Tendermint mette a disposizione dei nodi validatori (validators), identificati dalla loro chiave pubblica, e ogni nodo è responsabile del mantenimento di una copia integrale dello stato del sistema, della proposta di nuovi blocchi e del voto per validare i suddetti blocchi. A ogni blocco viene assegnato un indice incrementale (height), così facendo si avrà un blocco valido per ogni height. Ogni blocco viene proposto da un nodo diverso ogni volta (il nodo viene detto proposer), dividendo il processo di consenso in veri e propri round. Il processo di consenso può essere diviso in 3 fasi:

• Proposta (proposal): il proposer di turno propone un nuovo blocco e gli altri validatori lo ricevono. Se non lo ricevono entro un determinato periodo di tempo si passa al proposer successivo;

• Votazione (votes): la fase di votazione si suddivide anch'essa in due sotto parti, ossia pre-vote e pre-commit.

• Lock: Tendermint si assicura che nessun validatore inserisca più di un blocco a un dato indice (height).

Ogni round inizia con una nuova proposta. Il nodo che effettua la proposta prende le transazioni presenti all'interno della sua cache, chiamata Mempool, assembla il blocco e lo spedisce sulla rete tramite un messaggio firmato (ProposalMsg). Una volta che la proposta viene ricevuta da un nodo validatore, quest'ultimo firma un messaggio per il pre-vote di quella proposta e lo invia a tutta la rete. Se un validatore non riceve una proposta entro un determinato periodo di tempo (ProposalTimeout), il suo voto verrà considerato come nullo. Se almeno i 2/3 della rete hanno votato a favore del blocco, si passa a un'altra votazione, ossia quella per la pre-commit. In sintesi, la votazione di pre-vote prepara la rete a ricevere un nuovo blocco da inserire alla blockchain. Se la rete è pronta a ricevere questo nuovo blocco, ossia è stato votato dai 2/3 della rete, allora si passa alla votazione per il pre-commit e se un validatore riceve voti da almeno i 2/3 dei nodi, il blocco viene aggiunto alla blockchain e viene computato il nuovo stato del sistema.

L'altra componente di Tendermint, l'Application BlockChain Interface, funge da interfaccia tra il processo applicativo e il processo di consenso. L'ABCI comunica con il Core di Tendermint principalmente attraverso 3 tipi di messaggi: **DeliverTx, CheckTx e Commit**. Il messaggio **DeliverTx** viene usato ogni qual volta viene inviata una transazione sulla blockchain. L'applicazione poi dovrà verificare la validità della transazione rispetto allo stato attuale. Se la transazione risulterà valida, allora l'applicazione dovrà aggiornare il proprio stato con le nuove informazioni ottenute dalla transazione. Quando un'applicazione presente sulla rete Tendermint vuole inviare una transazione, i

dati immessi dall'utente vengono pre-processati e temporaneamente salvati all'interno di una memoria cache, la Mempool Cache. Prima di essere immessa nella Mempool vera e propria, ossia la memoria da cui poi un nodo può recuperare le transazioni da includere nel blocco che proporrà, il nodo verifica la validità della transazione tramite CheckTx: il contenuto della transazione viene confrontato con l'attuale stato del sistema e, se viene ritenuto coerente con esso, la transazione verrà accettata dal sistema, altrimenti verrà rifiutata. Di primo acchito può sembrare che CheckTx sia una DeliverTx semplificata, ma in realtà questi due messaggi vengono inviati in momenti diversi. Occorre analizzare le connessioni che l'interfaccia ABCI mantiene per capire la differenza tra i due.

L'applicazione ABCI presente su ogni nodo mantiene tre connessioni con il Tendermint Core: la Mempool Connection, usata per validare le transazioni presenti sulla Mempool tramite l'utilizzo di CheckTX, la Consensus Connection, usata soltanto quando viene effettuata la commit di un nuovo blocco, e la Query Connection, usata per effettuare query all'applicazione senza passare dal consenso. Lo stato dell'applicazione fornisce le informazioni necessarie, solo in lettura, alla Mempool Connection e alla Query Connection, mentre la scrittura viene presa in carico dalla Consensus Connection ed è proprio il blocco ricevuto da questa connessione a contenere tanti messaggi DeliverTX quante transazioni sono presenti nel blocco. Riassumendo, il messaggio CheckTx viene utilizzato quando una transazione deve essere inserita all'interno della Mempool, quindi prima del processo di consenso, mentre il messaggio DeliverTx viene usato dal consenso quando va ad assemblare il blocco, ordinando le transazioni. Infine, il messaggio Commit viene utilizzato per computare l'hash del Merkle tree corrispondente allo stato dell'applicazione.

EOS

EOS nasce da alcune celebrità del mondo crypto. Daniel Larimer è noto infatti per aver creato Bitshares e Steem, la blockchain che sta alla base del social Steemit. Larimer aveva già sperimentato su questi progetti un nuovo meccanismo di consenso di sua invenzione, il Delegated Proof-of-Stake (DPoS), meccanismo che brilla solo con il sorgere di Eos.

Fondata nel 2017 da Larimer e Brenden Blumer, Block.One è la società privata che ha sviluppato il protocollo Eos.io, famosa per aver lanciato la più grande ICO della storia: ha infatti raccolto oltre $ 4 miliardi, vendendo 1 miliardo di EOS nel corso del primo anno.

EOS mira ad essere l'infrastruttura più potente per lo sviluppo di Dapp in termini di:

- Scalabilità

- numero di transazioni al secondo
- interoperabilità con altre blockchain

Eos.io si è ispirato a Ethereum, la sua tecnologia tenta di risolvere i problemi storici legati alle Dapp, le quali hanno persino congestionato una blockchain longeva e stabile quale Ethereum, con conseguenti problemi di prestazioni per tutti gli utenti.

Diversamente dalle altre:

- Consente più transazioni al secondo (4.000)
- Ha eliminato completamente le commissioni addebitate agli utenti che effettuano transazioni. Queste vengono trasferite agli sviluppatori Dapp che devono pagare le risorse di rete.
- Consente agli sviluppatori di Dapp di usare linguaggi di programmazione tradizionali, invece che con un linguaggio specifico del progetto. Ad esempio, ci sono tre tipi di risorse che alimentano la blockchain di Eos.io che sono molto familiari a chi possiede nozioni basilari d'informatica:
    - Larghezza di banda (Disk): necessaria per la trasmissione delle informazioni attraverso la rete.
    - Calcolo (CPU): la potenza di elaborazione necessaria per eseguire un Dapp.
    - State Storage (RAM): utilizzato per memorizzare i dati sulla sua blockchain.
    - Utilizza la cripto valuta nativa EOS per votare gli aggiornamenti software e pagare i costi di gestione della sua blockchain.

Per creare e lanciare nuove Dapp, gli sviluppatori devono utilizzare gli Smart Contract. Combinazioni di più Smart Contract consentono via via di creare Dapp più complesse. Il gioco di compravendita Upland è una Dapp sviluppata su EOS, così come EOS Dynasty, un gioco di ruolo in cui le azioni dei partecipanti sono alimentate dalla cripto valuta.

Ciò nonostante nell'ultimo trimestre dell'anno, il network di EOS è divenuto fortemente congestionato, registrando un calo crescente degli utilizzatori.

TRON

Tron è una piattaforma digitale decentralizzata basata su blockchain con una propria cripto valuta, chiamata TRX. Fondata nel 2017 da un'organizzazione no-profit di Singapore, la Tron Foundation, Tron mira a ospitare un sistema di intrattenimento globale per la condivisione conveniente di contenuti digitali.

Il protocollo TRON, gestito principalmente dalla Fondazione TRON, distribuisce equamente le risorse di elaborazione tra i titolari di TRX con meccanismi di tariffazione interni come larghezza di banda ed energia. TRON fornisce una macchina virtuale decentralizzata, che può eseguire un programma utilizzando una rete internazionale di nodi pubblici. La rete ha zero commissioni di transazione e conduce circa 2.000 transazioni al secondo.

Le implementazioni di TRON richiedono commissioni di transazione minime per impedire agli utenti malintenzionati di eseguire attacchi DDoS gratuitamente. A questo proposito, EOS.IO e TRON sono abbastanza simili, a causa delle commissioni trascurabili, delle elevate transazioni al secondo e dell'elevata affidabilità, e come tali sono considerati una nuova generazione di sistemi blockchain. Alcuni ricercatori hanno definito TRON come un clone di Ethereum, senza differenze fondamentali. Il tasso di transazioni al secondo sulla blockchain di Tron è stato messo in discussione perché era molto al di sotto della sua pretesa teorica. Inoltre, un test di sicurezza informatica, nel maggio 2019, ha rivelato che un solo computer avrebbe potuto arrestare l'intera blockchain di TRON, attraverso un attacco DDoS (Distributed Denial of Service).

Come realizzare una blockchain da zero

In questo paragrafo riportiamo la descrizione ed i passaggi per costruire una semplice blockchain di criptovaluta, chiamata **thecoin**, usando Node.js.

Node.js è uno dei framework JavaScript più rivoluzionari dell'ultimo decennio, in quanto permette di utilizzare V8, l'interprete JavaScript di Google. Questo consente agli sviluppatori di realizzare web application con JavaScript non più solo lato client, ma anche sfruttandolo come linguaggio di programmazione lato server. La caratteristica principale di Node.js risiede nella possibilità che offre di accedere alle risorse del sistema operativo in modalità event-driven e non sfruttando il classico modello basato su processi o thread concorrenti, utilizzato dai classici web server.

Il modello event-driven, o “programmazione ad eventi”, si basa su un concetto piuttosto semplice: si lancia una azione quando accade qualcosa. Ogni azione quindi risulta asincrona a differenza dei pattern di programmazione più comune in cui una azione succede ad un’altra solo dopo che essa è stata completata. Ciò dovrebbe garantire una certa efficienza delle applicazioni grazie ad un sistema di callback gestito a basso livello dal runtime.

L’efficienza dipenderebbe dal considerare che le azioni tipicamente effettuate riguardano il networking, ambito nel quale capita spesso di lanciare richieste e di rimanere in attesa di risposte che arrivano con tempi che, paragonati ai tempi del sistema operativo, sembrano ere geologiche.

Grazie al comportamento asincrono, durante le attese di una certa azione il runtime può gestire qualcos’altro che ha a che fare con la logica applicativa, ad esempio. Di seguito la logica dei due approcci:

```
/** approccio sincrono (classico) **/

var dato = ottieniDatoDaRemoto(url);

alert(dato);

/** approccio ad eventi (asincrono) **/

ottieniDatoDaRemoto(url, function(dato) {

  alert(dato);

}); //la funzione ritorna subito
```

Mentre la prima parte di codice l’esecuzione aspetta la ricezione dei dati prima di effettuare l’alert, nel secondo caso il dato viene passato sotto forma di funzione callback. L’engine in questo caso si comporta mettendo in pausa la callback in attesa che la funzione esterna ottenga il dato ma non bloccando il resto dell’applicazione.

Dopo aver installato Node.js sulla propria macchina, andiamo a creare l’app nella cartella di lavoro che chiameremo **thecoin.js.**

Nella stessa cartella di sviluppo, installiamo la crypto libreria che andremo ad utilizzare, utilizzando il comando:

```
npm install --save crypto-js
```

Useremo questa libreria per importare tutti i moduli necessari per il nostro progetto.

Creiamo innanzitutto una classe BlockCypto come mostrato di seguito:

```
const SHA256 = require('crypto-js/sha256');
class BlockCypto{
   constructor(index, current_time, info, nextHash=" "){
   this.index = index;
   this.current_time = current_time;
   this.info = info;
   this.nextHash = nextHash;
   this.hash = this.computeHash();
   }
   computeHash(){
      return SHA256(this.info + this.nextHash + this.current_time +
JSON.stringify(this.info)).toString();
   }
}
```

Nel costruttore della classe che abbiamo creato, inizializziamo le sue proprietà e gli assegniamo i parametri, come segue:

- **crypto-js/sha256**: Questo è il modulo che abbiamo importato per calcolare l'hash di ogni blocco. Lo abbiamo convertito in stringa utilizzando il metodo toString() poiché il modulo restituirà l'oggetto.
- **index:** Questo è un numero distintivo che traccia l'indice di ogni blocco nella blockchain.
- **current_time**: Come dice il nome, tiene traccia dell'ora in cui ogni transazione viene completata.
- **info**: Tutti i dati delle transazioni completate vengono registrati e archiviati con questo metodo.
- **nexthash**: punta all'hash_key del blocco successivo nella catena di rete. Viene utilizzato principalmente per mantenere l'integrità della blockchain.

- **computeHash**: In base alle proprietà passate a questo metodo, viene utilizzato per calcolare l'hashkey del blocco successivo nella catena.

Una volta definito un singolo blocco, occorre definire la blockchain, ovvero il registro distribuito che mantiene la concatenazione dei singoli blocchi.

```
class Blockchain{
    constructor(){
        this.block1chain = [this.initGenesisBlock()];
    }
    initGenesisBlock(){
        return new BlockCrypto(0, "24/08/2021", "Initial Block in the Chain", "0");
    }
    latestBlock(){
        return this.block1chain[this.block1chain.length - 1];
    }
    addNewBlock(newBlock){
        newBlock.nextHash = this.latestBlock().hash;
        newBlock.hash = newBlock.computeHash();
        this.block1chain.push(newBlock);
    }
}
```

Come al solito, abbiamo il nostro costruttore che istanzia la blockchain.

Ma questa volta, abbiamo passato al costruttore il metodo initGenesisBlock(), che inizializza il blocco nella catena. La catena nel nostro caso si riferisce a un array di blocchi.

- **initGenesisBlock():** è quindi il primo blocco creato nella rete peer-to-peer e non è stato collegato a nessun altro. L'indicizzazione del primo blocco è posto a 0. Anche il primo nodo è stato creato usando la classe BlockCrypto creata in precedenza, passandogli tutti i parametri come argomenti.
- **latestBlock**: come dice la stessa parola, viene usato per trovare l'ultimo blocco aggiunto nella catena (la testa dell'array nel nostro caso). Come spiegato in precedenza, viene utilizzato per garantire l'integrità della catena in quanto l'hash del blocco corrente mappa l'hash del blocco precedente.

- **addNewBlock**: metodo che serve ad aggiungere un nuovo blocco alla catena. L'hash del blocco precedente viene abbinato all'hash del blocco corrente per impedire la manomissione della catena.

La caratteristica principale di una blockchain è che una volta che un blocco è stato aggiunto alla rete, non può essere modificato senza invalidare l'intera integrità della blockchain. Per eseguire ciò, utilizziamo la sicurezza digitale o l'hash crittografico, che garantisce la protezione e la convalida della blockchain producendo un nuovo hash ogni volta che viene apportata una modifica nel blocco. Eseguendo un loop sull'intera blockchain è possibile verificare se qualche hash è stato manomesso, tenendo conto dell'eccezione del primo blocco, che è hardcoded.

Inoltre, all'interno dello stesso metodo si andrà a verificare se le chiavi crittografiche di ogni due blocchi in serie puntano l'una verso l'altra. Se l'integrità della blockchain è stata compromessa, restituisce false; in caso contrario, nel caso non si riscontrino anomalie, ritorna true.

Creeremo questo metodo all'interno della classe Blockchain:

```
checkValidity(){
   // Checking validity
   for(let i = 1; i < this.block1chain.length; i++) {
      const currentBlock = this.block1chain[i];
      const nextBlock= this.blockchain[i-1];
   // Checking current blcok hash

   if(currentBlock.hash !== currentBlock.computeHash()) {
      return false;
   }
   // Comparing current block hash with the next block

   if(currentBlock.nextHash !== nextBlock.hash) {
      return false;
   }
   return true;
}
```

Possiamo quindi passare al testing del codice, creiamo una nuova istanza della classe Blockchain che chiamiamo thecoin, e aggiungiamo alcuni blocchi nella blockchain usando valori casuali.

```
let thecoin = new Blockchain();

thecoin.addNewBlock(new BlockCrypto(1, "24/08/2021", {sender: "Emilio Greco", recipient: "Franco Cicirelli", quantity: 20}));

thecoin.addNewBlock(new BlockCrypto(2, "24/08/2021", {sender: "Franco Cicirelli", recipient: "Emilio Greco", quantity: 349}));

console.log(JSON.stringify(thecoin, null, 4));
```

Digitando questo comando nel nostro terminale: node thecoin.js si otterrà il listato delle tre transizioni. Realizzare applicazioni con Node.js come abbiamo visto è facile. La potenza di questa piattaforma, una vasta raccolta di librerie e la relativa semplicità d'uso ne hanno incoraggiato la rapida diffusione. Tuttavia, dato che un'applicazione sviluppata con Node.js è composta da codice JavaScript che viene eseguito a runtime da Node (o più precisamente, dal motore V8 di Crome), è necessario disporre dell'intero ambiente per poterla eseguire correttamente. Tale caratteristica può complicare la distribuzione dell'applicazione. Non si può infatti pretendere che ogni macchina target disponga di Node.js e di tutte le librerie usate dall'applicazione. Una soluzione al problema, estremamente diffusa in ambito backend, consiste nel distribuire l'applicazione sotto forma di immagine Docker.

Per realizzare un'immagine Docker in grado di eseguire la nostra applicazione, bisogna prima di tutto creare un file che la descriva: il Dockerfile. Esso è un semplice file di testo che indica a Docker quale immagine di base utilizzare, quali file copiare nell'immagine, quali porte esporre e quale processo eseguire all'interno del container.

```
FROM node:10-alpine
MAINTAINER vostro nome
RUN mkdir -p /home/node/app/node_modules
RUN chown -R node:node /home/node/app
WORKDIR /home/node/app
COPY package*.json ./
RUN npm install
```

```
COPY . .
RUN chown -R node:node /home/node/app
USER node
EXPOSE 3000
CMD [ "node", " thecoin.js" ]
```

L'immagine che verrà generata sarà basata sulla *node:10-alpine*: si tratta di un'immagine ufficiale di Node.js, basata sulla distribuzione Linux *Alpine* e contenente Node.js alla versione. Le istruzioni seguenti copiano il contenuto della directory di lavoro nell'immagine, inclusi i moduli richiesti e assegnano all'utente "node" la proprietà della directory contenente l'applicazione. Infine, l'istruzione CMD indica l'entrypoint del container, ovvero il comando da eseguire. In questo caso ovviamente, verrà eseguito il comando node thecoin.js.

Esistono altre immagini di base che si possono utilizzare laddove, ad esempio, l'applicazione dipenda da librerie native non disponibili in Alpine. Il vantaggio di questa immagine di base rispetto ad altre è che è molto più "snella", e questo permette di creare immagini leggere. L'immagine descritta nel Dockerfile dell'esempio occuperà poche decine di MB.

Per creare l'immagine, sarà sufficiente utilizzare il comando **Docker build**, seguito dal tag da assegnare all'immagine. Ad esempio:

```
# docker build -t miaapp:latest
```

Dapp stato dell'arte

Un recente sondaggio ha rivelato che la maggior parte dei progetti Dapp è stata avviata nel 2018 ed è autofinanziata. Il 72% dei progetti interpellati è stato avviato nel 2018, con la maggior parte dei team con una media di 2-5 sviluppatori. L'87% dei progetti è costruito sulla blockchain di Ethereum, seguono EOS (19%) e Tron (8%).

Il 59% delle Dapp è stato autofinanziato, mentre il 16% è stato finanziato da vendite di token. Il 33% delle Dapp non dispone di un piano per scalare nel caso l'aumento del volume delle transazioni causi dei colli di bottiglia. Il **numero limitato di utenti crittografici** (67%) e la **inefficace user experience della crittografia** (44%) sono stati indicati dagli sviluppatori come principali punti critici.

Circa il **50% dei progetti** utilizza un **backend cloud centralizzato e strumenti centralizzati come Infura** per connettersi alla blockchain di Ethereum. La maggior parte dei progetti, prosegue il report, monetizza attraverso le commissioni sulle transazioni. Il più grande ostacolo alla maggiore adozione delle Dapp è il processo di onboarding dei nuovi utenti, secondo il 75% degli intervistati.

Oggi il mondo delle applicazioni (o Apps) è molto presente nella nostra vita così che se chiedete al più imbranato nella tecnologia che cosa sono Whatsapp, Instagram, Twitter o Glovo saprebbe risponderti che sono Apps. Un'app è un'applicazione informatica progettata per essere eseguita su smartphone, tablet e PC e permettono di eseguire compiti di qualsiasi tipo (inviare e-mail, scrivere in una chat, effettuare videochiamate, giocare un videogioco ecc).

Forse non ci hai mai pensato, ma le Apps presentano un problema comune: sono centralizzate. Ciò significa che se un giorno i proprietari di Whatsapp volessero mettere un prezzo di abbonamento per utilizzare la famosa app di messaggistica, questa potrebbe essere usata solo pagando. Quando parliamo di Apps centralizzate, ci riferiamo ad applicazioni che dipendono da un'entità centrale che ha il potere di decidere qualsiasi cosa, senza la necessità di prendere in considerazione i suoi utenti.

Questa premessa ha senso quando si tratta di capire **che è una Dapp, in che cosa si differenzia dalle Apps e a che cosa serve**.

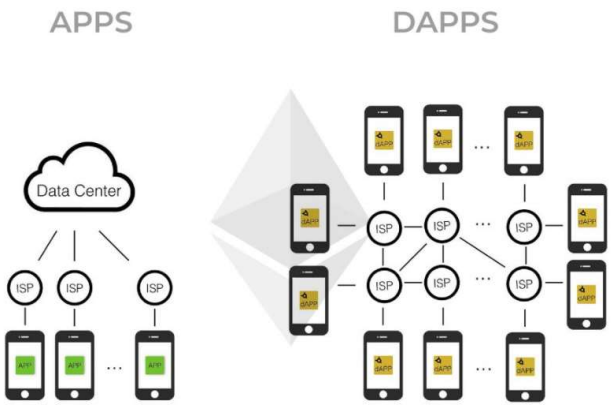
Una Dapp è un'applicazione decentralizzata, ossia un'app che non dipende da un ente centrale, ma dipende dalla stessa comunità di utenti che la utilizza e consente l'interazione diretta tra utenti e fornitori.

Una Dapp è essenzialmente una ***applicazione open source in esecuzione su una rete decentralizzata da pari a pari*** (peer-to-peer), di cui nessuna entità ha il controllo completo e le informazioni sono continuamente condivise tra i partecipanti.

Tra le caratteristiche principali di una Dapp figurano le seguenti:

- **Open Source (codice sorgente aperto)**. Nelle applicazioni a codice chiuso, gli utenti devono fare affidamento sugli sviluppatori delle applicazioni in quanto non possono accedere direttamente ai loro dati. Nelle Dapp tutte le modifiche devono essere decise dal consenso (la maggior parte) dei suoi utenti.
- **Consenso Decentralizzato**. Tutti i registri del funzionamento dell'applicazione sono memorizzati in una blockchain pubblica e decentralizzata. In una Dapp, anche le transazioni sono trattate attraverso un meccanismo di consenso. Quando la maggior parte dei nodi approva la transazione, questa viene elaborata.

- **Incentivato**. I validatori della blockchain sono ricompensati con token di cripto valute.
- **Protocollo**. La comunità dell'applicazione deve concordare un algoritmo crittografico per dimostrarne il valore. Ad esempio, Bitcoin utilizza Proof of Work (PoW) e Ethereum attualmente utilizza PoS sebbene in futuro abbiano pianificato un PoW/Proof of Stake ibrido (PoS).
- **Non esiste un punto centrale di guasto**: Le Dapp consentono di decentralizzare i dati in esse memorizzati tra tutti i loro nodi, che sono indipendenti l'uno dall'altro. In caso di guasto di un nodo, gli altri nodi non saranno interessati. Dal momento che le Dapp sono distribuite e non dipendono da un singolo server, non esiste un punto centrale di **guasto**.



La blockchain di Ethereum è la blockchain principale su cui gli sviluppatori creano Dapp e il suo white paper definisce tre categorie principali di Dapp:

- **Applicazioni che gestiscono denaro**: Si tratta di Dapp che generalmente hanno la propria blockchain e che forniscono agli utenti metodi per gestire le proprie finanze e denaro. Un esempio è Bitcoin, che fornisce agli utenti un sistema di monetizzazione distribuito e decentralizzato in cui gli stessi utenti sono i proprietari dei loro soldi in queste app.
- **Applicazioni semi finanziarie di Blockchain**: Si tratta di applicazioni che integrano il denaro con eventi esterni nel mondo reale, al di fuori della blockchain. Un esempio in questa categoria sono le ICO (Initial Coin Offering), un meccanismo di raccolta fondi simile all'IPO con l'unica differenza che vi è la partecipazione di cripto valute al posto del denaro fiat.
- **Organizzazioni autonome decentrate (DAOs)**: Questa terza categoria di Dapp utilizza tutte le caratteristiche dei sistemi decentrati e distribuiti. Sono il tipo più popolare di Dapp e un esempio di queste Dapp sono le applicazioni per il voto online o la governance decentrata.

Le startup Dapp, costruite su piattaforme di blockchain, costituiscono oggi la maggior parte delle ICOs. La maggior parte delle Dapp di categoria 2 e 3 utilizza la piattaforma Ethereum.

Sebbene Ethereum sia attualmente la piattaforma più scelta dagli sviluppatori di Dapp, Cardano, QTUM e NEO sono anche molto popolari per lo sviluppo di Dapp.

Un limite riscontrato, durante la diffusione delle Dapp, ma più in generale con la sempre più ampia diffusione della blockchain è la mancata possibilità di sfruttare dati esterni al mondo blockchain e quindi off-chain, rimanendo circoscritti alle sole informazioni del mondo on-chain. Ad esempio, se si vuole realizzare una Dapp per la gestione di scommesse si ha bisogno di un ente esterno che comunichi agli Smart Contract di riferimento della Dapp i risultati esatti dei match affinché le scommesse vengano pagate correttamente. Proprio da questa esigenza di avere a disposizione negli Smart Contract i dati relativi al mondo off-chain che si è introdotto il concetto di oracolo.

Algoritmi di swarm intelligence distribuiti

In questo capitolo descriveremo a lunghe linee due noti algoritmi di swarm intelligence distribuiti, analizzando la possibilità di esecuzione su una rete Blockchain of Things. Nel primo paragrafo viene descritta l'esecuzione di un algoritmo di clustering su una rete "tradizionale" di nodi IoT interconnessi attraverso una rete MQTT e ne viene suggerita una possibile implementazione attraverso una Dapp realizzata su una **Application-Specific blockchain** facendo utilizzo di Tendermint Core. Allo stesso modo, a seguire viene illustrato il noto algoritmo PSO ed una sua possibile applicazione attraverso Dapp utilizzando il framework Tendermint.

Edge Clustering on MQTT

Gli animali sociali o gli insetti in natura spesso mostrano una forma di comportamento collettivo emergente noto come " Flocking ". Il modello Flocking è un modello di calcolo bio-ispirato per simulare l'animazione di uno stormo di entità. Rappresenta il gruppo in movimento come si vede negli stormi di uccelli e nei banchi di pesci in natura. In questo modello, ciascun individuo prende le sue decisioni di movimento da solo in base a un piccolo numero di regole semplici e reagisce agli stimoli che sente dai suoi vicini e nell'ambiente. Queste semplici regole locali di ogni individuo generano un comportamento globale complesso dell'intero stormo. Il grado di località è determinato dalla portata della sensibilità del singolo individuo. Un individuo non reagisce agli

stimoli al di fuori del suo raggio di sensibilità e determina il suo movimento basandosi solo sulle informazioni locali. Le tre regole di Reynolds sono sufficienti per riprodurre i comportamenti naturali di un gruppo. Nel modello MSF (Multiple Species Flocking), oltre a queste tre regole di azione di base si introduce una quarta regola, quella di similarità. In base a questa regola, un boid in uno stormo cerca di stare vicino a quegli individui che hanno caratteristiche simili alle sue (stessa specie) e stare lontano da altri che invece hanno caratteristiche dissimili, producendo in questo caso dei raggruppamenti in base al valore di somiglianza. Le prime due regole di Reynolds richiedono di conoscere **la posizione dei vicini**, mentre la terza richiede di conoscere la loro **velocità**. Infine la quarta regola necessita di conoscere oltre la posizione, anche il **vettore caratteristico** dei vicini al fine di calcolare il valore di somiglianza. Possiamo in definitiva utilizzare due piani bidimensionali: il piano di volo (che cambia ad ogni iterazione) ed il piano delle somiglianze (che rimane invariato per tutto il processo).

Se andiamo a considerare il seguente esempio:

- fissata la distanza **d** di sensibilità dell'algoritmo pari a 4;
- dato un insieme di test costituito da 4 cluster ben distinti come quello mostrato in figura

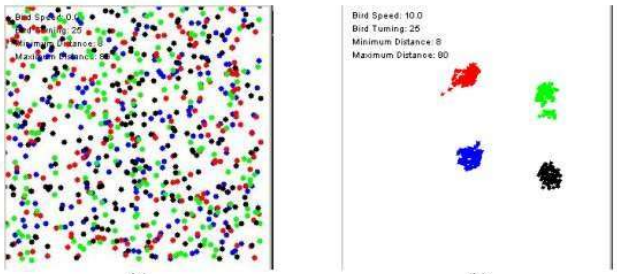
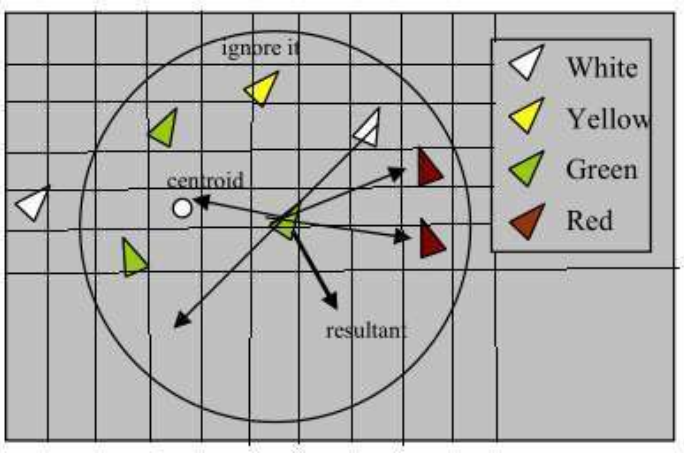


Fig. 2. Multiple species bird flocking simulation

Avremo che ad ogni iterazione, ogni boid, come ad esempio boid centrale rappresentato in figura



dovrà calcolare la propria velocità e posizione come combinazione lineare delle quattro risultanti determinate dalle quattro regole MSF.

$$v = w_{sr}.v_{sr} + w_{ar}.v_{ar} + w_{cr}.v_{cr} + w_{ds}.v_{ds} \quad . \qquad (5)$$

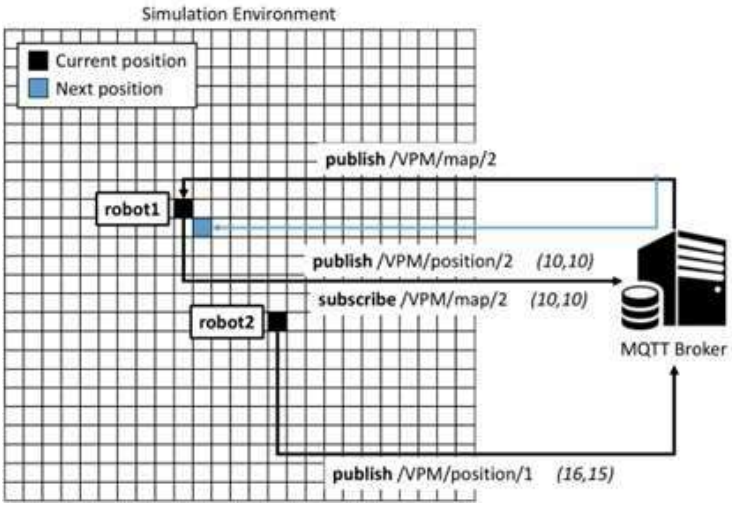
Un boid viene implementato solitamente come tre vettori di double, ovvero vengono rappresentati su unpiano di volo di punti continui.

```
private Vector position; //Current Position of Boid

private Vector velocity; //Current Velocity of Boid

private Vector acceleration; //Acceleration acquired by a boid in each unit oftime
```

discretizzando il piano di volo e suddividendolo in celle sarà possibile associare ad ogni cella un "topic mqtt". Pertanto i calcoli delle regole di Reynolds dovranno essere discretizzate in modo che diano come risultato unaposizione della griglia come mostrato nell'immagine successiva (costituito da un certo numero di pixel sufficienti per la visualizzazione).



Realizzata la griglia di volo e distribuiti i boid in modo casuale all'interno di essa, il prossimo passo è definire come effettuare il calcolo del vicinato.

Per il calcolo del vicinato si possono utilizzare due approcci:

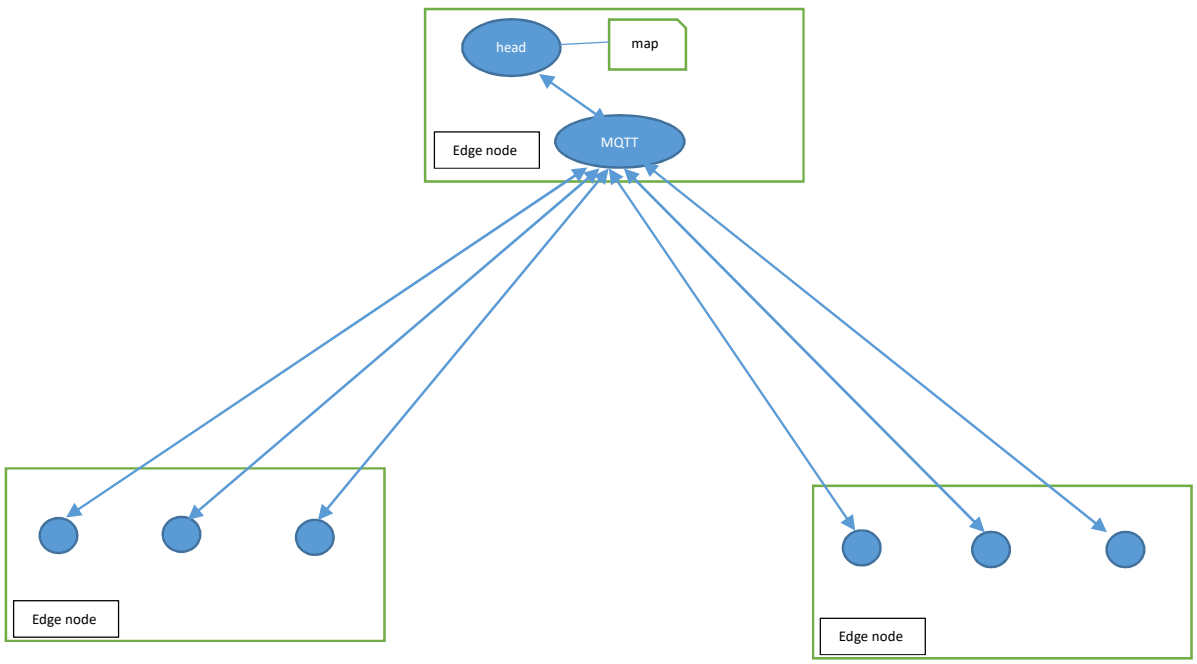
1. viene calcolato da un agente posizionato sul mqtt broker (approccio centralizziamo)
2. Ogni agente ricava da solo i propri vicini (approccio distribuito)

Nel primo caso, deve esistere un agente supervisore, che tra le altre cose visualizza l'evoluzione del sistema, calcola le metriche e segue il calcolo del vicinato. Affinché possa fungere da coordinatore, esso deve essere sottoscritto a tutti i topic della griglia.

Non solo occorre definire chi elabora il vicinato, ma anche come. Anche in questo caso esistono due approcci:

1. Il coordinatore aspetta di ricevere tutte le posizioni, in modo da avere un quadro completo del piano di volo, dopo di che calcola e comunica a tutti la lista dei nuovi vicini. In questo caso si crea un punto di sincronizzazione delle attività. Il problema in questo caso è se un boid si disconnette o si blocca, bloccando l'esecuzione del processo.
2. Il coordinatore elabora continuamente il vicinato in maniera asincrona, mandando gli aggiornamenti in tempo reale. In questo caso, ogni boid potrebbe essere aggiornato su uno stato del vicinato in maniera errata perché nel frattempo i vicini si sono mossi di posizione. Questo è molto probabile viste le velocità di elaborazione diverse tra i nodi, alla gestione delle richieste non simultanea del supervisore, alle latenze di rete.

Supponendo di voler realizzare la soluzione centralizzata. Ogni boid, che occupa la cella della griglia, calcola la sua posizione al prossimo step, la comunica al broker, il quale calcola i vicini sulla mappa e a sua volta inoltra le informazioni solo ai boid che hanno subito una variazione del vicinato.



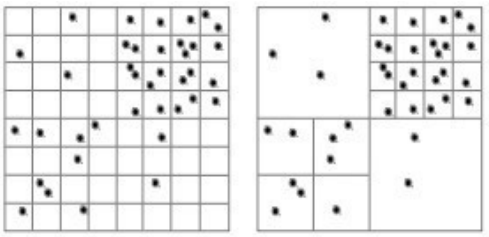
La griglia evolve in maniera sincrona seguendo la sequenza di questi passi:

elabora posizione->comunica posizione-> ricevi lista vicini -> elabora posizione.

Ovvero ogni agente è un automa a tre stati. Il sincronismo è dovuto al fatto che per calcolare i vicini di un punto x, abbiamo la necessità di ricevere tutte le nuove posizioni raggiunte dei boids adiacenti. Ogni boid pubblica solo sul topic "broker" dove è in ascolto solo il supervisore. Il supervisore pubblicherà sui topic "boid1","boid2",etc. Infine il boidx sarà in ascolto solo sul topic "boidx".

Sebbene ogni boid esegue il calcolo della sua posizione sul piano di volo, parallelizzando il calcolo, questo vantaggio vene perso a causa della sincronizzazione sul nodo coordinatore.

È possibile migliorare il parallelismo creando un'infrastruttura dinamica che supporta la pubblicazione/sottoscrizione spaziale scalabile (noto anche come application-layer multicast for Adaptive load balancing), implementando un Progressive partitioning come in figura:



Il sistema deve essere supportato da un insieme di broker locali che gestiscono la messaggistica su aree di interesse in maniera indipendente. I boid sono quindi assegnati a broker specifici in base alle loro posizioni più recenti e vengono riassegnati (roaming) in modo trasparente ad altri broker se oltrepassano i confini regionali. In un VE con partizioni spaziali, le interazioni sono limitate a quelle all'interno della regione corrente o a parte delle regioni vicine confinanti (l'interazione tra regioni dipende dal raggio di sensibilità dei boids (AoI) e dal numero di boid che si trovano a confine). Gli attori interagiscono solo con le partizioni che si sovrappongono alle loro AoI.

Il partizionamento spaziale statico soffre di due grandi limitazioni:

1. Per prima cosa, la ricerca di un partizionamento ottimale è una operazione non semplice: una grande dimensione delle partizioni può provocare una mancata corrispondenza della mappatura degli AoI con le regioni, causando sovraccarico aggiuntivo di messaggi irrilevanti per i boid; mentre un partizionamento con dimensioni piccole delle regioni si

tradurrà in un numero eccessivo di regioni, inducendo più carico per operazioni di roaming e di messaggi inter regione.

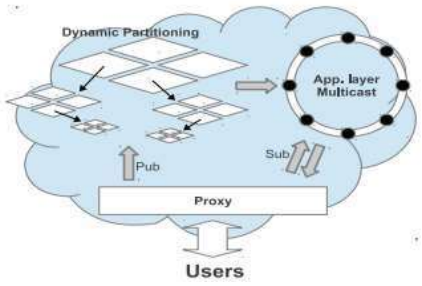
2. Se utilizziamo una partizione statica (ad esempio, calcolata offline utilizzando una funzione hash), questo potrebbe portare alla generazione di hotspot, cioè partizioni ad un'elevata densità che rallenterebbero l'evoluzione del sistema.

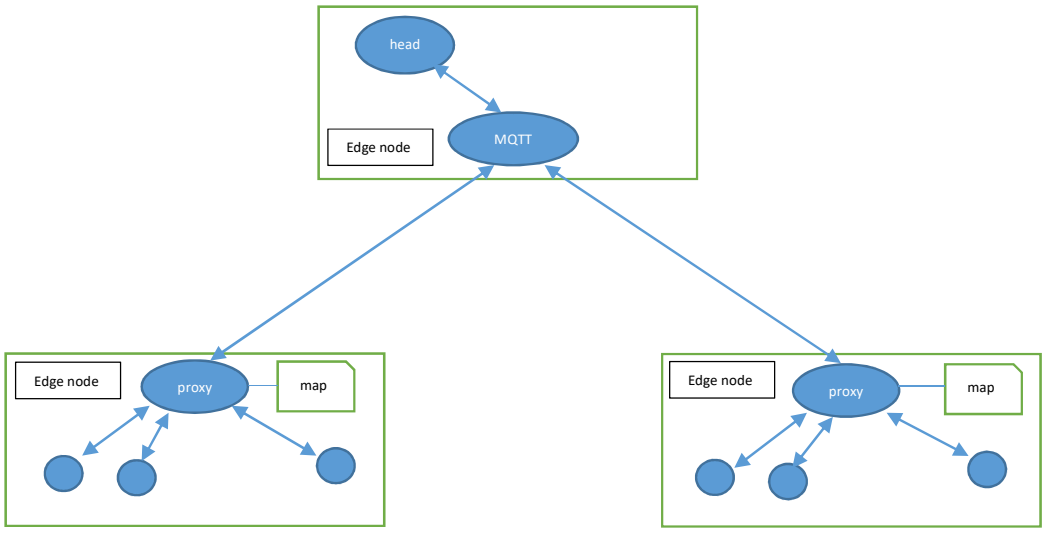
Il partizionamento progressivo supera queste limitazioni. Le regioni a partizione progressiva sono suddivise in base alla densità di elementi al loro interno. Se l'attività in una regione supera una soglia, il partizionamento viene affinato suddividendo la partizione originale in partizioni più piccole. Al contrario, se si rilevano partizioni inattive (al di sotto di una soglia), queste vengono unite per ridurre il numero totale di regioni. Ciò fa sì che il numero di regioni sia proporzionale alla distribuzione e frequenza degli eventi. Un algoritmo noto per il partizionamento progressivo è il quad-tree.

Il quad-tree è' una rappresentazione gerarchica ad albero. Per pubblicare un messaggio nel VE, un host trova le partizioni più adatte alla pubblicazione e inoltra l'evento a loro. La radice della gerarchia racchiude tutto il VE. La pubblicazione di un evento richiede la ricerca di un elenco di partizioni che intersecano l'area della pubblicazione.

Esistono tuttavia due principali fonti di incoerenza in un VE distribuito. Le prime sono le incongruenze negli stati VE, ad esempio quando due boid tentano di occupare la stessa posizione nel caso di boid di confine. In secondo luogo, le incongruenze di infrastruttura causate da operazioni conflittuali da parte dell'host a causa di presupposti errati. Ad esempio, se si considera lo scenario in cui un proxy utilizza una versione obsoleta della mappatura delle partizioni.

Per gestire il roaming occorre implementare un provider di transazioni distribuito. Questo ci consente di combinare la pubblicazione/sottoscrizione (pub/sub) con query spaziali. La piattaforma/sistema distribuito che supporta questo meccanismo è composto da tre componenti: proxy, partitioning e multicast, come rappresentato in figura:





Ogni componente è completamente distribuito sui nodi di calcolo. In questo modo nessun nodo funge da coordinatore e ne è necessario mantenere alcuna forma di stato globale. Il proxy è il frontend del sistema e si occupa dell'interazione dei boid con la restante parte dell'infrastruttura di comunicazione.

Il secondo approccio che andremo ad analizzare utilizza la conoscenza indiretta del vicinato e non fa affidamento ad un nodo coordinatore ne locale e ne globale. La struttura a griglia dell'environment consentirebbe ad ogni boid di essere notificato solo dalle celle che rientrano nel suo campo di azione (AoI)ed occupate da altri boid. Se ogni cella può contenere un solo boid e se supponiamo che non esistono problemi di concorrenza perché la velocità e la posizione di ogni boid è regolata dalle regole di Reynolds che ne impediscono la collisione, allora un agente che si sposta nella prossima cella (x,y), prima di farlo, si sottoscrive ai seguenti topic:

tutte le posizioni (a,b) per x-d<=a=<x+d e y-d<=b=<y+d

Queste posizioni rappresentano il suo campo di azione. Eseguita la sottoscrizione, pubblica la sua nuova posizione sul topic (x,y) , ad esempio col messaggio "cella x-y occupata!".

Il messaggio mqtt che viene pubblicato, deve essere un messaggio **di tipo reteined**, in modo che chi si sottoscriverà alla cella anche dopo la pubblicazione del messaggio, verrà notificato dell'ultimo stato ricevuto. Chi è registrato sul topic (x,y) riceverà la notifica che la cella è stata occupata con un tipo di comunicazione **multicast**.

Ma questo vantaggio è solo apparente perché per far si che ciò avvenga occorrono una serie di messaggi "di configurazione" iniziali e finali, necessari per ridefinire le relazioni di vicinato. Ma quanti messaggi sono necessari per riconfigurare la rete delle conoscenze?

Occorre inviare i seguenti messaggi seguendo questa sequenza:

a) Per i vecchi followers occorre inviare tanti messaggi di **unsubscribe** quante erano le celle limitrofe della vecchia posizione. Se non cancelliamo la nostra vecchia rete di conoscenza, continueremo a ricevere messaggi provenienti da loro.

b) Bisogna mandare un messaggio vuoto (**public**) non reteined alla cella che lasciamo, per cambiare il suo stato. NOTA: un public innesca a sua volta tanti messaggi per quanti sono i subscribe della cella notificata.

c) Per i nuovi followers: Ogni boid deve inviare tanti messaggi di **subscribe** quante sono le celle limitrofe alla nuova posizione raggiunta (compresa la cella che è stata appena lasciata), in modo da essere notificato se arriva un nuovo vicino ad occupare una di quelle celle; NOTA: se le celle a cui ci sottoscriviamo hanno ricevuto come ultimo messaggio, un messaggio di tipo retained, riceveremo in automatico da questi il messaggio del loro ultimo contenuto.

d) Bisogna infine pubblicare il nuovo stato della cella che andremo ad occupare con un messaggio di tipo reteinded. Anche qui avremo una serie di messaggi di notifica verso i followers.

Se il numero di celle dell'area AoI del boid è pari ad N celle, in numero di notifiche che transiteranno per ogni movimento del singolo boid sarà:

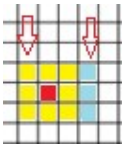
N (punto a) + N+1 (punto b) + N+N (punto c) + N+1 (punto d) = 5N+2 scambi di messaggi

Tuttavia è possibile ottimizzare eseguendo subscribe e unsubscibe solo verso le nuove celle che entreranno nell'area di interesse e quelle che ne usciranno. In questo caso avremo:

N+1 (punto b) + N+1 (punto d) + 2Y = 2N+2+2Y scambi di messaggi dove Y è il numero di celle lasciate nello spostamento.

Esempio:

sia AoI =1; ovvero N= 8 possibili vicini:



Punto a = 3 msg ; punto b= 9 msg; punto c= 3; punto d=9 msg

In questo esempio per il movimento di un boid occorrono 24 messaggi per riconfigurare il vicinato.

Se abbiamo 10 boind su una griglia di 50 posizioni, avremo 240 messaggi per ogni transizione. Se consideriamo invece una mappa centralizzata dei vicini o una distribuita con comunicazione broadcast, avremo 10x10 = 100 messaggi.

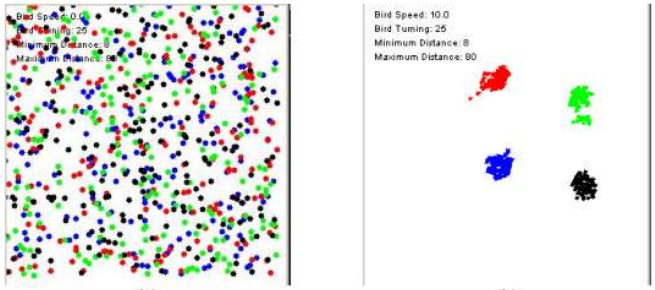


Fig. 2. Multiple species bird flocking simulation

Se consideriamo l'esempio mostrato sopra, nella condizione iniziale di volo (immagine a sinistra) avremo più del doppio dei messaggi di una comunicazione broadcast tra boids. Nello stato finale, relativamente all'esempio sopra riportato avremo: 2N msg (dove N è all'incirca un terzo dell'area di volo), nel caso migliore avremo un numero di messaggi uguale al modello di comunicazione broadcast.

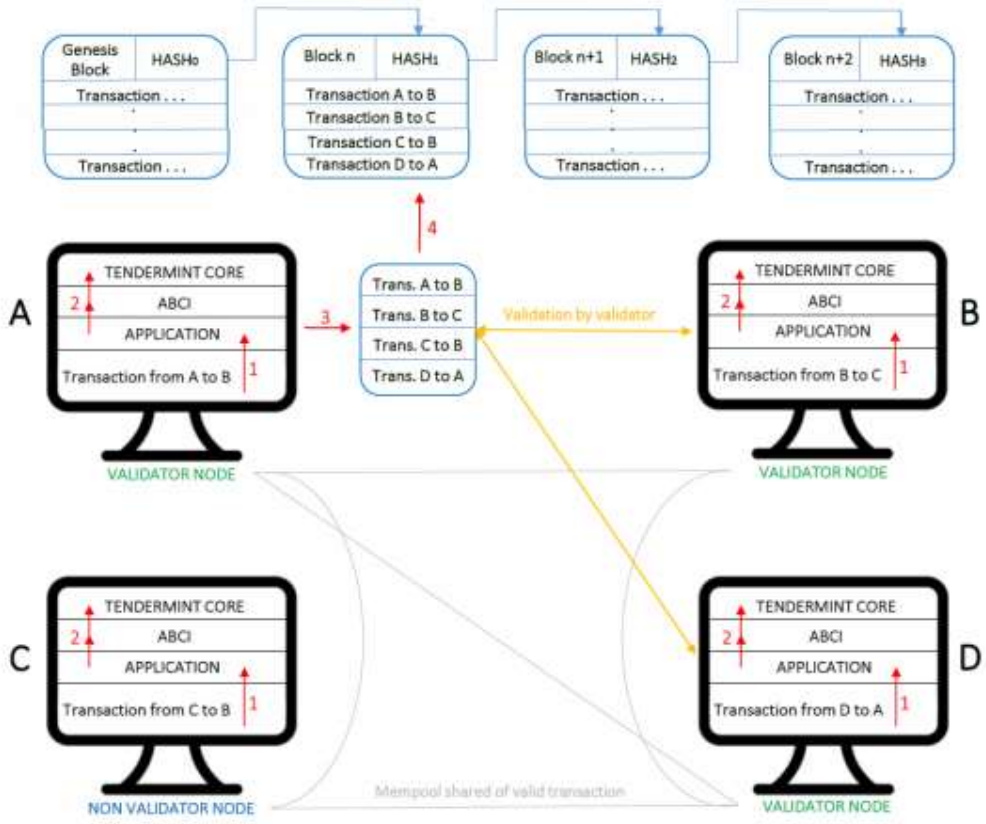
Edge Clustering on Blockchain

Nel paragrafo precedente abbiamo fatto una disamina delle possibili soluzioni per realizzare un algoritmo di clustering basato su flocking, poggiandoci su una infrastruttura ed un protocollo di comunicazione di largo uso nel campo IoT, come MQTT. Le problematiche emerse sono di diversa natura: la gestione della temporizzazione, della concorrenza, la gestione delle regioni, del vicinato, piuttosto che la completa vulnerabilità ad attacchi di nodi malevoli, etc.

Nell'introduzione a questo elaborato abbiamo evidenziato tutti questi aspetti ed abbiamo visto come un approccio che fa uso di una infrastruttura blockchain, può in molti casi, superare queste problematiche. Tuttavia tra i vari tipi di blockchain esaminate, anche se quasi tutte offrono strumenti di programmazione Turing completi, sono state pensate al fine di sviluppare e gestire cripto valute e transizioni economiche e quindi più adatte ad **eventi una tantum,** sono quindi poco adatte alla creazione di piattaforme decentralizzate complesse. A questo scopo, per queste finalità specifiche, occorre quindi realizzare una blockchain ad oh, o più precisamente una **application-specific blockchain.**

Tendermint è progettato per essere facile da usare, semplice da capire, altamente performante e utile per un'ampia varietà di applicazioni distribuite.

La figura seguente mostra un diagramma semplificato del funzionamento di Tendermint.



Il funzionamento del sistema è abbastanza semplice, ogni volta che un utente decide di eseguire una transazione, la inviata attraverso l'App a sua disposizione (azione 1 in figura) al middleware che ne esegue la convalida. Il risultato della transizione è un messaggio restituito da Tendermint Core in merito alla validità o non validità della transazione (azione 2 in figura). Se la transazione è valida, sarà inclusa nel mempool (raccolta di transizioni) dei nodi validatori. Quello eseguito è solo il primo step del processo di validazione. A questo punto, il nodo validatore che ha eseguito la prima convalida, propone la transizione, assieme alle altre contenute nel suo mempool. Esegue cioè la proposta di inserimento di un nuovo blocco nella chain, inviandolo una richiesta agli altri nodi validatori della rete (azione 3 in figura) e, a seguire se ottiene il consenso di almeno 2/3 di essi, il nuovo blocco viene inserito nella blockchain (azione 4 in figura). L'inserimento nella blockchain avviene quindi solo se più di 2/3 dei nodi validatori eseguono il pre-commit dello stesso blocco nello stesso turno. Se ciò non avviene, il blocco passa al turno successivo.

L’applicazione ABCI, che definisce di fatto la struttura della blockchain da realizzare, può essere arbitrariamente complessa, scritta per ogni stack software e può avere un proprio database per memorizzare e gestire il proprio stato. Al di sopra dell’applicazione ABCI troviamo il software applicativo (DApp). Utilizzando la blockchain (Tendermint Core + ABCI) come una API qualsiasi, è possibile definire un registro condiviso rappresentante il piano di volo dei singoli boid. L’API provvederà a replicare ogni aggiornamento del piano di volo a tutti i boid interessati.

Seguendo lo schema descritto nella figura precedente, ogni boid del ClusterFlock realizza un elemento della rete o se vogliamo un client della Dapp. Quindi ad ogni turno di esecuzione il boid esegue una query sulla blockchain ed estrae le transizioni eseguite dai boids al turno precedente. Ogni transazione contiene l’informazione della posizione e velocità del boid sul piano di volo, nonché “il colore” che identifica il cluster di appartenenza.

I dati estratti vengono elaborati dalla Dapp e viene generata una nuova transizione con l’inserimento della nuova posizione del boid sul piano di volo. L’avvio del processo di ClusterFlock genera la blockchain di archiviazione, ma ne determina anche l’eliminazione a conclusione dell’esecuzione, in modo da liberare risorse di memoria. Tendermint offre la possibilità di lavorare con le regioni ed implementare meccanismi di publish/subscribe, per cui è possibile rendere l’algoritmo scalabile a piacimento.

Parallel Particle Swarm Optimization

Le prime implementazioni dell’algoritmo PSO parallelo sono stati sviluppate per macchine multiprocessore utilizzando la strategia master-slave. In questa strategia, il processore master coordina tutti i passaggi dell'algoritmo e gestisce l’esecuzione delle singole funzioni obiettivo sui processori slaves. L'algoritmo sincrono parallelo di PSO, può essere visto come una semplice estensione dell'algoritmo seriale. È sostanzialmente identico all'algoritmo seriale, con la differenza che le valutazioni della funzione obiettivo vengono eseguite in parallelo su più processori slave.

L'algoritmo sincrono mostra un punto debole nella sua strategia: la necessità di sincronizzazione, rende l'algoritmo limitato dallo slave più lento. In altre parole, al punto di sincronizzazione il master deve attendere che tutti gli slave terminino il proprio lavoro per continuare l'algoritmo lasciando in attesa gli eventuali slave disponibili. Per superare questa limitazione è stata implementata una strategia asincrona di parallelizzazione, dove non esiste punto di sincronizzazione. Viene così sostituito il concetto di volo delle particelle con la definizione di pseudo-volo. Negli algoritmi

sincroni, il numero di valutazioni della funzione fitness è uguale al numero di particelle, mentre nell'algoritmo asincrono questo numero non è costante perché non si attende che tutte le particelle abbiano concluso la loro esecuzione per eseguire una valutazione della convergenza. Pertanto indipendentemente da quali particelle hanno effettuato queste valutazioni, il master verifica la permanenza ottimale e i criteri di arresto senza interrompere il lavoro svolto dagli slave. In conseguenza, l'unica perdita di tempo nella parallelizzazione si verifica quando uno slave vuole comunicare con il master durante l'esecuzione della fase di ottimizzazione. In questa situazione, lo slave diventa momentaneamente inattivo.

Per ottenere l'asincronia, l'esecuzione delle particelle sono poste in una coda FIFO (First In First Out). Poiché il tempo di calcolo della funzione obiettivo varia, e con esso varia anche l'ordine delle particelle in coda per essere schedulate su un processore, di conseguenza alcune particelle potrebbero non contribuire all'ottimizzazione in uno pseudo-volo. Inoltre, gli slave possono eseguire un numero diverso di valutazioni, il che rende l'algoritmo asincrono incapace di mantenere la consistenza. Un approccio sincrono mantiene la coerenza tra implementazioni sequenziali e parallele, evitando così l'alterazione delle caratteristiche di convergenza dell'algoritmo.

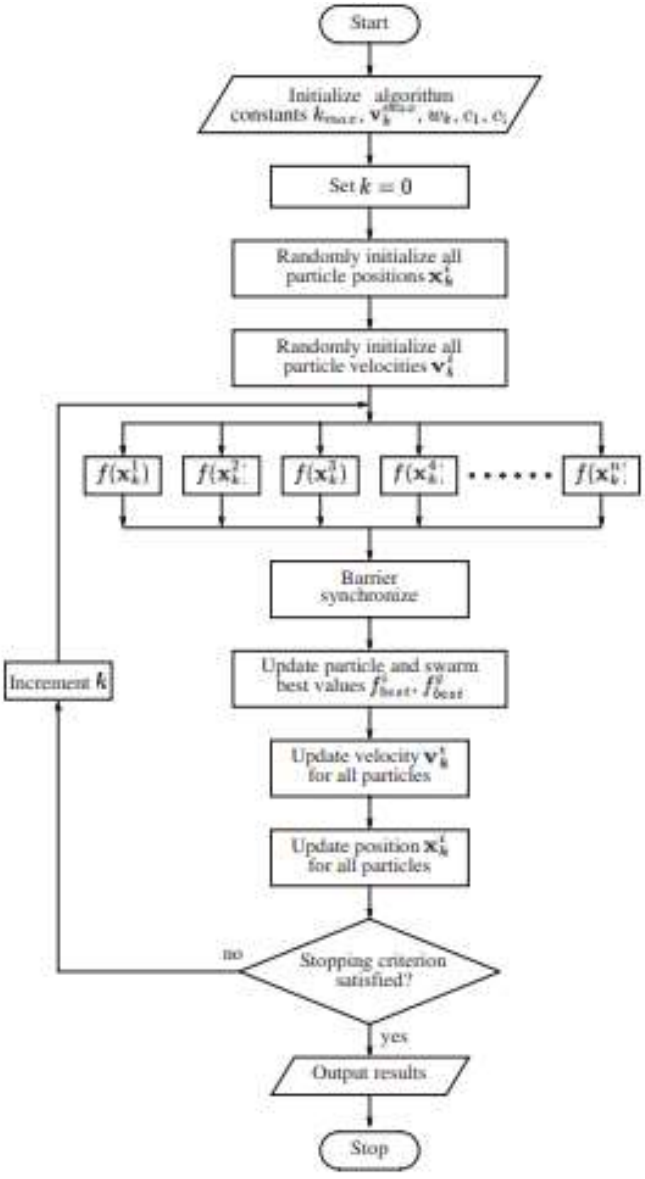
Poiché i vettori di velocità e posizione aggiornati della particella i sono calcolati utilizzando la posizione migliore PiK e la posizione migliore a livello globale PGK fino all'iterazione k , l'ordine delle valutazioni della funzione particella influenza l'esito dell'ottimizzazione. Di conseguenza, se permettiamo all'ordine delle particelle di cambiare continuamente a seconda della velocità con cui ogni processore li elabora, vengono alterate le caratteristiche di convergenza.

Un'implementazione del PSO asincrona parallela è presente in letteratura ed è stata sviluppata utilizzando l'approccio multi-agente. L'approccio multi-agente può generalmente gestire un problema suddividendolo in sotto problemi più semplici, in modo che gli agenti si devono dedicare solo a sotto-attività del problema generale. Nella programmazione, l'uso dell'approccio multi-agente è utilizzato per realizzare il parallelismo è l'esecuzione simultanea di calcoli (eventualmente collegati) al fine di accelerare l'elaborazione di problemi informatici intensivi e per eseguire un numero elevato di operazioni in un tempo limitato.

I sistemi multi-agente (MAS) possono essere caratterizzati dalla presenza di un'entità centrale denominata "broker", con cui tutti gli agenti del sistema possono comunicare e funge da "portalettere" verso gli altri agenti del gruppo. Ogni agente, prima di poter interagire col sistema, deve dichiararsi al Broker e deve fornirgli le sue caratteristiche.

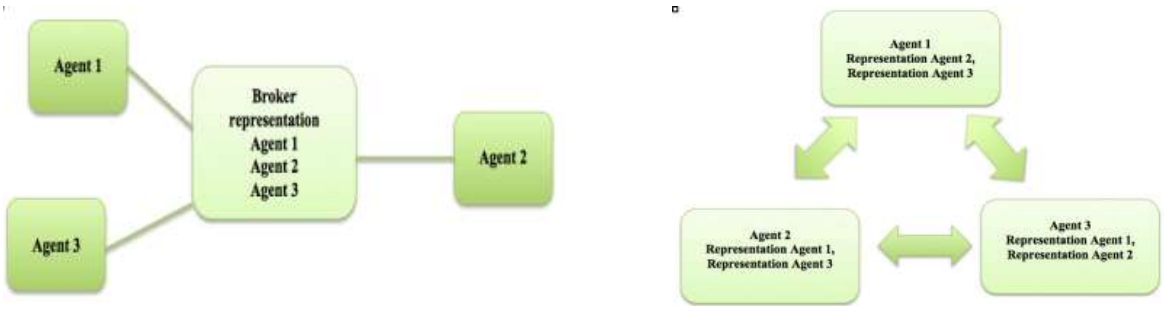
Il vantaggio di architettura centralizzata di comunicazione consiste nella facilità di aggiunta e rimozione di un agente; perché un nuovo agente si integri, è sufficiente stabilire un protocollo di interazione con il Broker. Lo svantaggio è che la centralizzazione può rallentare gli scambi e la comunicazione nel sistema.

Di seguito viene mostrato l'algoritmo di un Parallel PSO sincrono:



In un MAS che utilizza un approccio di comunicazione distribuito, ogni agente ha l'autonomia di interagire con gli altri agenti senza intermediario. Ogni elemento deve mantenere una base di conoscenza che descrive le caratteristiche e gli indirizzi degli agenti con cui collabora. Quando un agente vuole fornire un nuovo servizio, deve informare tutti gli altri di questo nuovo servizio. Il vantaggio dell'approccio distribuito è che facilita gli scambi e migliora la comunicazione. Lo

svantaggio è che la fornitura del nuovo servizio richiede un aggiornamento della base di conoscenza di ciascun agente.



Parallel Particle Swarm Optimization on BlockChain

L'implementazione dell'algoritmo su piattaforma blockchian è molto simile al sistema MAS basato su broker. La blockchain sostituisce il lavoro del brocker del sistema MAS, di fatti un nuovo nodo che vuole aggiungersi alla chain, deve dichiararsi ed utilizzare i formalismi della chain a cui si sottoscrive; ogni nodo della chain non conosce il numero o la collocazione degli altri nodi ed interagisce semplicemente con la chain.

Rispetto all'algoritmo PSO parallelo sincrono, nel nostro caso non abbiamo il blocco Master che controlla il flusso dell'applicazione e sincronizza i processi. In uno scenario in cui abbiamo N processi identici che eseguono le stesse operazioni in parallelo, quindi occorre parallelizzare l'intero flusso del programma mostrato nel paragrafo precedente. Per l'implementazione dell'algoritmo si può decidere per entrambe le versioni: asincrona o sincrona. La versione asincrona è quella che viene più naturale con l'uso di una blockchain in quando l'unica informazione di scambio con lo swarm è il valore della Gbest finora calcolata. Al termine di ogni operazione di ottimizzazione, ogni elemento dello swarm aggiorna, se necessario, la propria local best ed eventualmente anche il global best, se ad esempio il valore del local best è migliore del global best finore calcolato dallo swarm. Successivamente aggiorna la sua velocità e posizione.

In questo caso ci potrebbero essere più di una particella a richiedere l'aggiornamento del valore della gBest. Questo aspetto però non ci preoccupa in quando tutte le transizioni vengono temporizzate dalla piattaforma, l'unica accortezza da rispettare nel caso ad esempio di utilizzo della piattaforma Tendermint è quella di utilizzare transazioni con conferma come: "*broadcast_tx_commit*". Se vogliamo possiamo usare questa transazione come punto di sincronismo con gli altri partecipanti allo swarm. ovvero il nodo prima di aggiornare la propria posizione e la propria velocità deve attendere

l’approvazione del nuovo global best. Qualora vogliamo mantenere un comportamento del tutto asincrono, ogni nodo comunica al chain il proprio gbest e continua nella sua elaborazione utilizzando però il gbest memorizzato nella chain (eseguendo una quesry reqest) per mantenere la consistenza dell’applicazione. Ogni nodo conterrà quindi una sequenza di richiesta di aggiornamento della global best ed il valore della gbest allo step precedente. L’aggiornamento della global best avverrà attraverso la chiamata della funzione “**deliverTx**” (messaggio usato per l’inserimento del mempool nella chain). DeliverTx analizza tutto l’insieme delle transizioni (richieste di aggiornamento della gbest) ed aggiorna il gBest prendendo l’ottimo tra tutte le richieste.