Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica per il Management

APPROCCIO BLOCKCHAIN PER LA GESTIONE DEI DATI PERSONALI

Relatore:
Chiar.mo Prof.
STEFANO FERRETTI

Presentata da:
RICCARDO CAVALLIN

Correlatore:
Chiar.mo Dott.
MIRKO ZICHICHI

II Sessione
Anno Accademico 2019/2020

Introduzione

Il termine Blockchain è sicuramente una delle buzzword più in voga nel mondo informatico del giorno d'oggi. Le applicazioni di questa tecnologia, la più famosa delle quali è Bitcoin, sono innumerevoli e coinvolgono settori molto diversi fra loro. Le criptovalute ne fanno da padrone, ma la tracciabilità che si riesce ad ottenere con essa apre le porte a molti utilizzi come nell'ambito alimentare, con Walmart e Carrefour che già la utilizzano per monitorare il percorso di diversi alimenti. Tutte le attività che coinvolgono degli scambi possono beneficiarne, come ad esempio il commercio portuale oppure gli atti notarili. Secondo un report di MarketsandMarkets il mercato della blockchain raggiungerà il valore di quasi 40 miliardi di dollari per la fine del 2025 [1]. Una tecnologia in così rapida crescita può essere d'aiuto anche nel risolvere uno dei problemi più sentiti nella società "iperconnessa" in cui viviamo oggi: la protezione della propria identità online. Esistono già delle piattaforme che vogliono creare un web decentralizzato e restituire all'utente il controllo dei propri dati, come il progetto "Solid" del padre del World Wide Web Tim Berners Lee. L'utilizzo della blockchain per perseguire questo obiettivo è un possibilità molto affascinante che renderebbe i proprietari del dati più consapevoli del loro ruolo e del potere di cui dispongono. La gestione dei dati effettuata con questa modalità deve inoltre tenere in considerazione i regolamenti a tutela dei dati personali ed adottare tutte le precauzioni necessarie per risultare conforme. Nel presente elaborato è stato quindi progettato un Data Marketplace per la condivisione di dati fra utenti e aziende o altre entità interessate. Inoltre è stato costruito uno scenario sperimentale

per la valutazione dell'architettura di un Data Marketplace, in cui vengono confrontati diversi scenari per la memorizzazione di dati. La tesi in oggetto è quindi strutturata come segue:

- Nel primo capitolo viene presentato lo stato dell'arte della tecnologia blockchain e di qualche sua particolare implementazione. Inoltre si discute del nuovo regolamento europeo sulla Protezione dei Dati Personali e le sue implicazioni per l'architettura trattata. Infine vengono descritte diverse soluzioni per l'archiviazione dei dati in rete a supporto della blockchain.

- Nel secondo capitolo viene presentata una demo di un Data Marketplace realizzato utilizzando la blockchain Ethereum. Viene anche spiegato il concetto fondamentale di Smart Contract in relazione allo scambio di dati.

- Il terzo capitolo contiene delle analisi prestazionali effettuate su tre diversi servizi di archiviazione di file online, utilizzati per lo storage dei dati personali. I provider sono IPFS, Google Drive e Amazon Web Services.

- Il quarto e ultimo capitolo contiene le conclusioni e qualche considerazione sulle difficoltà riscontrate e possibili sviluppi futuri.

Indice

Elenco delle figure

Capitolo 1

Stato dell'arte

1.1 La Blockchain

Una blockchain (letteralmente "catena di blocchi") è una struttura dati pubblica e decentralizzata che fa parte della famiglia delle Distributed Ledger Technologies. Le DLT si basano su un registro distribuito, non necessariamente sottoforma di catena, accessibile e modificabile da tutti i nodi che fanno parte di una rete. Lo scopo principale è quello di avere un sistema sicuro per effettuare transazioni fra i partecipanti alla rete senza bisogno di una entità centrale regolatrice che supervisiona ogni operazione. Il sistema è di conseguenza più difficilmente attaccabile dall'esterno perché gli utenti si accordano in autonomia riguardo agli eventi, evitando quindi di avere un "single point of failure". In una blockchain ogni blocco contiene un insieme di transazioni avvenute in un certo arco temporale ed è connesso agli altri attraverso un riferimento al blocco precedente della catena. La crittografia è un componente fondamentale, infatti il collegamento fra blocchi è ottenuto grazie all'hash del blocco precedente che viene salvato in quello successivo. Per poter mantenere la coerenza fra i nodi esiste un meccanismo di consenso distribuito. Un nuovo blocco per poter entrare nel registro dev'essere approvato da un determinato numero di nodi e successivamente viene inviato un messaggio a tutti gli altri peer che aggiorneranno il proprio registro

includendo il nuovo blocco.

1.1.1 Bitcoin

La nascita della prima blockchain risale al 2008, quando Satoshi Nakamoto pubblicò il whitepaper *"Bitcoin: A Peer-to-Peer Electronic Cash System"* [4]. Al giorno d'oggi Bitcoin rimane l'applicazione più famosa e di successo della tecnologia blockchain. L'idea di Nakamoto è quella di creare una moneta alternativa a quella fisica che ne corregga alcuni difetti. Infatti la moneta legale presenta qualche svantaggio come il fatto che può essere facilmente rubata, è difficile da monitorare e da tassare in quanto semi-anonima. Il punto focale tuttavia è la presenza obbligatoria di un'autorità centrale che stampa e controlla la moneta. È necessario quindi avere fiducia verso questa autorità ed espone inoltre a rischi riguardanti la sicurezza del sistema perché si ha un unico punto di rottura. Bitcoin invece utilizza un sistema completamente decentralizzato che si appoggia su una rete peer-to-peer e un registro pubblico di transazioni (la blockchain). Interessante è il sistema escogitato per aggiungere un nuovo blocco alla catena, tutti i peer infatti devono essere concordi nell'aggiungere il nuovo blocco. Il consenso viene ottenuto attraverso un particolare algoritmo denominato Proof of Work che consiste nella produzione di un valore particolare che assicura senza alcun dubbio che del lavoro computazionale è stato effettuato. Coloro che concorrono a generare questo digest devono risolvere una funzione crittografica, la cui difficoltà viene aggiustata in continuazione in modo da creare un nuovo blocco ogni 10 minuti circa. I miners devono mettere a disposizione una notevole potenza di calcolo perché il processo di calcolo procede per tentativi ed errori, riducendo in media tempi per la validazione del blocco se si dispone di molta capacità di calcolo. Proprio da qui nasce uno dei problemi di Bitcoin cioè l'altissimo consumo di energia richiesta per la PoW. Si pensi che il consumo energetico annuo associato a Bitcoin è più alto di quello di paesi come la Svizzera e Hong Kong. Confrontandolo con altri circuiti elettronici si scopre che una transazione Bitcoin consuma più del triplo dell'energia richiesta da 100000

transazioni VISA [5]. Il processo di mining viene incentivato attraverso delle "fee" (mance) che vengono rilasciate ad ogni nuova transazione verso coloro che validano il blocco. Questa tassa è a tutti gli effetti il costo delle operazioni. I micropagamenti tuttavia sono penalizzati da questo sistema perché spesso la tassa è più alta dell'importo da trasferire. Nonostante ciò la complessità di calcolo è anche un punto a favore in quanto protegge il network dal problema della doppia spesa. Infatti se un nodo malevolo volesse spendere più fondi di quanti in realtà ne possiede dovrebbe modificare un blocco, generando quindi una variazione nel suo codice hash. Ne consegue che dovrebbe ricalcolare a catena anche tutti i blocchi successivi e per farlo dovrebbe disporre di una potenza di calcolo incredibilmente alta, almeno maggiore di quella di tutti gli altri nodi messi assieme. I blocchi sono quindi immutabili e le transazioni non possono essere modificate a posteriori. Il successo di Bitcoin ha portato alla nascita di diverse tecnologie simili che vogliono esportare il concetto di blockchain in altri ambiti. Non più solamente monete virtuali ma applicazioni in moltissimi altri ambiti: dall'agroalimentare fino all'IoT.

1.1.2 Ethereum

Nel 2013 Vitalik Buterin presenta un nuovo tipo di blockchain programmabile chiamata Ethereum [6]. Il progetto è open source e consiste in un tentativo di decentralizzazione di qualsiasi informazione presente su Internet. Quella che oggi viene chiamata blockchain 2.0 è un linguaggio Turing-completo e una piattaforma *general-purpose*. La struttura di base è sempre un insieme di nodi connessi attraverso una rete peer to peer ma con la possibilità di eseguire dei programmi open source. Creando degli Smart Contract infatti si possono scrivere applicazioni decentralizzate di qualsiasi natura con delle specifiche regole, tipi di operazioni e stati delle transazioni. Un esempio reale di Smart Contract verrà presentato nella sezione 2.3.1. Ethereum ha anche la sua criptovaluta chiamata Ether (ETH). In Ethereum esistono degli oggetti chiamati account che possono modificare lo stato del sistema. Sono identificati da indirizzi di 20 byte e si suddividono in due tipi: *Externally*

Owned Account e *Contract Account*. Un EOA contiene al suo interno un "nonce" cioè un contatore per assicurare che ogni transazione venga eseguita una sola volta, il suo indirizzo e il saldo di ETH. Un CA invece possiede in aggiunta del codice e uno spazio dati persistente su cui effettuare le operazioni. I primi quindi non hanno codice ma sono i responsabili della creazione e firma delle transazioni, mentre i secondi attivano il loro codice solamente quando ricevono un messaggio. I messaggi in Ethereum possono contenere dei dati e possono essere oggetto di risposte da parte di un Contract Account. Le transazioni invece sono dei pacchetti di dati firmati inviati da un Externally Owned Account contenenti i messaggi, un ammontare di dati da spedire e due valori chiamati STARTGAS e GASPRICE. Il rischio nell'eseguire del codice è quello di imbattersi in un loop infinito, perciò la contromisura adottata è quella di attribuire un costo ad ogni computazione. Questo prezzo è il GASPRICE, mentre STARTGAS è il limite massimo di step computazionali che la transazione può eseguire. Nel caso in cui il gas non sia sufficiente vengono ripristinati tutti gli stati iniziali ad esclusione delle tasse pagate per l'esecuzione fino a quel momento, mentre se vi è del gas rimanente viene restituito al proprietario.

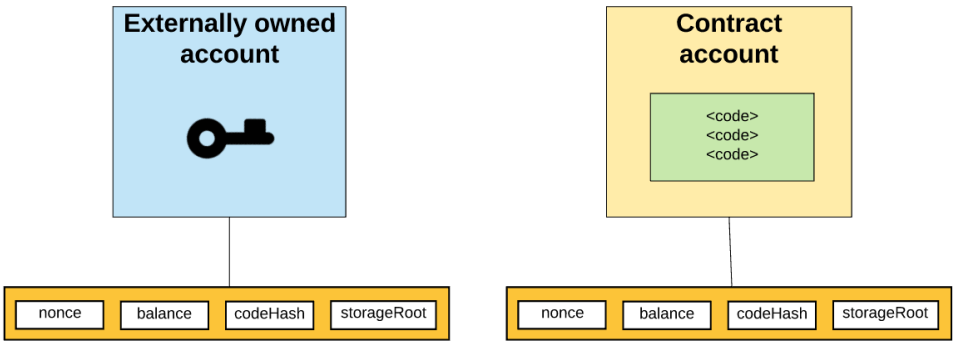

Figura 1.1: I due tipi di account su Ethereum

Come in Bitcoin l'algoritmo di consenso utilizzato è Proof of Work. Ne

consegue che per validare un nuovo blocco è necessario effettuare della computazione e che i miner vengono ripagati con una fee (in questo caso sono ETH ovviamente). Tuttavia esiste un altro modo per ripagare i miner: nel caso in cui eseguano il codice dello Smart Contract. Per ogni computazione effettuata essi riceveranno un compenso.

1.1.3 Il problema della scalabilità

La scalabilità è definita come la capacità da parte di un sistema di essere facilmente modificabile per supportare una quantità di lavoro crescente. La diffusione delle blockchain è stata limitata proprio da questo fattore. Eseguire una transazione alla volta e sostenere una grossa quantità di calcolo per approvarle significa ottenere un collo di bottiglia per l'intero network. Il numero di transazioni al secondo che attualmente l'architettura Bitcoin riesce a supportare oscilla fra 3.3 e 7. Oltre alla dimensione massima limitata del blocco (4 megabytes) l'altra causa è il tempo che trascorre fra l'aggiunta di un blocco e l'altro nella catena. Una transazione richiede quindi in media 10 minuti per essere effettuata, quando circuiti come VISA processano 2000 transazioni al secondo con picchi di 56000 [7]. Anche Ethereum è limitata ad un basso numero di transazioni che si aggira sulle 7-15 al secondo [8]. Il problema è spesso citato come *scalability trilemma*. Secondo questo trilemma è impossibile avere un sistema decentralizzato, consistente (cioè in cui tutti i nodi hanno la stessa copia del registro e operano sugli stessi dati) e scalabile. Bisogna necessariamente scegliere due di queste tre proprietà sacrificandone la terza.

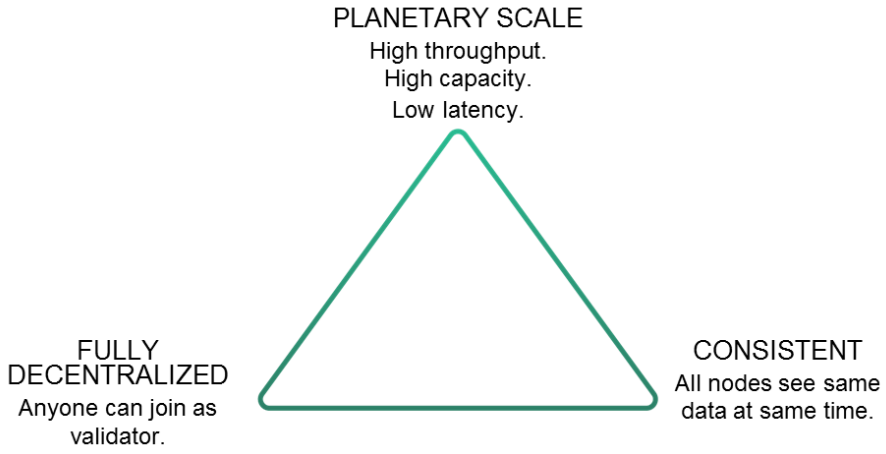

Figura 1.2: Il trilemma della scalabilità

Supportare un maggior numero di transazioni al secondo è un requisito fondamentale per raggiungere un maggior bacino di utenza, ed è per queste ragioni che sono nate alcune implementazioni particolari dei registri distribuiti.

IOTA

Il primo approccio significativo verso la ricerca della scalabilità è quello proposto da IOTA. Questa particolare DLT vuole creare una criptovaluta per l'Internet of Things. Per poter riuscire nel suo scopo l'architettura deve essere in primis scalabile per supportare l'enorme quantità di dispositivi connessi e poi avere dei bassi costi di transazione. IOTA propone quindi una struttura dati diversa dalla blockchain chiamata *Tangle* basata su un grafo direzionato aciclico (DAG) [9].

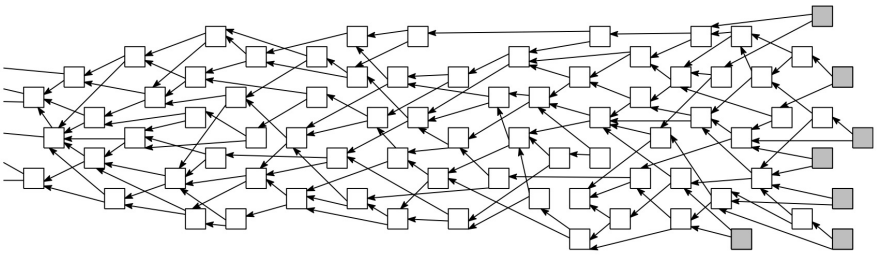

Figura 1.3: Il Tangle di IOTA

L'idea è che i blocchi non sono salvati sotto forma di catena ma sono sparsi come in un grafo e collegati gli uni agli altri con degli archi che costituiscono i riferimenti fra le diverse transazioni. Per poter aggiungere un nuovo blocco bisogna approvare due *tips*, cioè due transazioni che sono state aggiunte alla rete ma che non sono state ancora approvate. Questi tips vengono selezionati utilizzando l'algoritmo *Random Walk Monte Carlo*. La Proof of Work in questo caso consiste nel verificare se le due transazioni sono in conflitto e nel risolvere un puzzle crittografico. Questo problema matematico è meno oneroso a livello di risorse rispetto al caso Bitcoin perché costruito tenendo a mente la capacità di calcolo dei dispositivi IoT. In IOTA infatti non c'è distinzione fra user e miner e tutti i nodi possono partecipare al consenso. Non viene quindi richiesta nessuna fee per aggiungere nuovi blocchi, rendendo il sistema molto più adatto alle microtransazioni. Tuttavia proprio a causa del fatto che i dispositivi IoT non hanno grosse capacità di memoria il Tangle viene "potato" di tanto in tanto, salvando un'immagine del sistema e consentendo ai nodi di eliminare tutte le transazioni precedenti a quel momento. Questo procedimento va contro i principio di immutabilità e persistenza delle DLT. Per ovviare a ciò sarebbe possibile effettuare un partizionamento del registro (detto sharding) andando però incontro a possibili attacchi di doppia spesa. Un nodo potrebbe infatti effettuare due transazioni conflittuali in due shard diverse e la verifica sarebbe molto complicata. Inoltre in una DAG non esiste un sistema per avere un ordinamento temporale accurato delle transazioni e un attacco al network potrebbe avere successo con solo il 34% della

potenza di calcolo [10]. Per evitare di incorrere in comportamenti malevoli nelle prime fasi di vita della piattaforma IOTA ha introdotto un particolare nodo chiamato Coordinator che ha il compito di sorvegliare le transazioni. Ad ogni modo questa soluzione centralizza parzialmente la DLT e nonostante non sia più in uno stadio infantile vede ancora la presenza del nodo.

Radix DLT

Radix DLT è una DLT relativamente nuova nata appositamente per affrontare il problema della scalabilità fin dalla nascita. È stata accuratamente progettata dal principio per affrontare il problema ed ha annunciato di aver raggiunto l'incredibile soglia di 1 milione di transazioni durante un test in cui replicava 10 anni di transazioni Bitcoin [11]. Sebbene promettente Radix ha riscontrato, e sta riscontrando, notevoli problemi su più fronti. L'algoritmo con il quale è stato condotto il test, Tempo, è stato recentemente abbandonato a causa di problemi di sicurezza. Il suo sostituto Cerebrus [12] si basa comunque sullo stesso principio del precedente: lo sharding. Lo sharding è una procedura che consiste nella frammentazione di un database in porzioni più piccole. Operando su porzioni ristrette e veloci del registro distribuito si aumenta la scalabilità. Il Radix Ledger è diviso in un numero altissimo di shard, pari ad addirittura 18.4 quintilioni (2^{64}). Si pensi che se Google dovesse suddividere tutti i suoi dati in questa quantità di shard ognuno di essi occuperebbe solamente 4 bit. Il controllo double spending risulta semplice perché ad ogni dato è associato l'indirizzo del mittente della transazione. Radix vuole quindi raggiungere la scalabilità con il parallelismo cioè la possibilità di effettuare più transazioni contemporaneamente, al contrario di quanto avviene nelle blockchain classiche in cui i blocchi sono validati uno alla volta. Purtroppo allo stato attuale Radix non dispone di una rete globalmente accessibile ma solamente di una rete locale chiamata Betanet. Inoltre lo sviluppo di applicativi diventa particolarmente complicato a causa della mancanza di una community attiva e di un supporto adeguato.

1.2 Dati e GDPR

I dati costituiscono ormai il bene di maggior valore all'interno della società, essi vengono creati in continuazione e circolano con estrema facilità. Coloro che ne traggono beneficio sono sopratutto le aziende che possono effettuare una profilazione incredibilmente accurata avvalendosi dei social network, spesso senza che gli utenti ne siano consapevoli. Le grandi tech corporate infatti gestiscono tutti i dati generati dagli utenti con procedure non sempre trasparenti e i Data Leak sono sempre dietro l'angolo, come insegna il caso Cambridge Analytica [13] e il più recente Google Project Nightingale [14]. Lentamente le persone si stanno rendendo consapevoli dell'importanza delle loro azioni online e di come possono essere analizzate per influenzare i comportamenti.

1.2.1 General Data Protection Regulation

Il 25 maggio 2018 è diventato operativo il Regolamento Generale sulla Protezione dei Dati [15]. Dopo quattro anni di preparazione infatti l'Unione Europea ha riscritto la disciplina della Privacy a livello europeo. La necessità di emanare un nuovo regolamento in materia di privacy nasce dalla continua evoluzione degli stessi concetti di privacy e protezione dei dati personali e quindi della relativa tutela dovuta principalmente alla diffusione del progresso tecnologico. I dati personali a cui si fa riferimento sono quelli associati alle persone fisiche. Oggetto del regolamento sono anche il trattamento automatizzato dei dati personali e il trattamento non autorizzato di dati personali contenuti in un archivio. Le finalità ultime sono quelle di proteggere e tutelare meglio la protezione dei dati di tutti i cittadini in Europa e di armonizzare la normativa europea in materia di Privacy, eliminando le differenze di approccio fra Stati membri. Il dato personale è qualsiasi informazione riguardante una persona fisica identificata o identificabile. L'interessato, cioè il soggetto a cui appartiene il dato, può essere identificato direttamente con identificativi come ad esempio nome, indirizzo o identificativo online ma anche indiret-

tamente facendo riferimento ad elementi più caratteristici della sua identità fisica, fisiologica, genetica, psichica, economica, culturale o sociale. Di conseguenza è vietato trattare dati personali particolari, cioè quei dati personali che rivelino l'origine razziale o etnica, le opinioni politiche, le convinzioni religiose filosofiche, o l'appartenenza sindacale, nonché trattare dati genetici, dati biometrici intesi a identificare in modo univoco una persona fisica, dati relativi alla salute o alla vita sessuale o all'orientamento sessuale della persona. Il trattamento è invece inteso come qualsiasi operazione o insieme di operazioni di gestione del dato come la raccolta, conservazione, modifica, consultazione, comunicazione e cancellazione su supporti informatici ma anche cartacei o analogici. Il titolare del trattamento, cioè il responsabile dell'applicazione del GDPR, è tenuto a mettere in atto misure tecniche e organizzative adeguate per garantire e comprovare che il trattamento è conforme alla norma e per proteggere i dati fin dalla progettazione (by design) e per impostazione predefinita (by default). Il GDPR stabilisce inoltre i seguenti 7 principi:

- Leicità e correttezza: l'interessato deve essere consapevole ed esprimere il proprio consenso al trattamento con una decisione informata.
- Trasparenza: il titolare deve adottare misure appropriate per fornire informazioni e comunicazioni facilmente accessibili con un linguaggio semplice e chiaro.
- Limitazione della finalità: il trattamento deve essere effettuato dentro i limiti delle finalità del trattamento.
- Minimizzazione: il titolare del trattamento deve limitare la raccolta di informazioni personali a ciò che è direttamente rilevante e necessario a raggiungere uno scopo specifico. I dati inoltre devono essere conservati solo per il tempo necessario per raggiungere lo scopo.
- Esattezza: i dati raccolti devono essere esatti e se necessario deve essere possibile aggiornarli.

- Limitazione della conservazione: i dati non devono essere conservati per un periodo superiore a quello necessario per il raggiungimento dello scopo per cui erano stati trattati.

- Integrità e riservatezza: il trattamento deve garantire una sicurezza adeguata con misure tecniche ed organizzative per proteggere i dati da trattamenti non autorizzati o illeciti, dalla loro perdita o distruzione accidentale.

In aggiunta vengono riconosciuti dei diritti all'interessato come il diritto ad essere informato, il diritto accesso, rettifica e cancellazione dei dati (chiamato anche diritto all'oblio). Molto importanti sono anche il diritto alla limitazione del trattamento e portabilità dei dati. Quest'ultimo si traduce nella possibilità di aver accesso o scaricare i dati in un formato open in modo che siano comprensibili con l'ausilio di un calcolatore. Da sottolineare sono il diritto ad opporsi a determinate forme di trattamento e quello a non essere sottoposto a decisioni basate unicamente sul trattamento automatizzato dei dati.

1.2.2 Conseguenze per la blockchain

Ad una prima lettura del regolamento GDPR potrebbe sembrare in forte contrasto con le blockchain e il mondo delle DLT in generale. La natura delle DLT infatti è quella di un registro pubblico decentralizzato immutabile basato su una rete peer to peer. Il diritto alla trasparenza sarebbe quindi rispettato perché tutto ciò che accade è visibile ai nodi che fanno parte della rete. Lo stesso non si può dire riguardo alla limitazione della conservazione perché i dati all'interno di una rete peer to peer vengono conservati per sempre o almeno fino a quando almeno un nodo della rete ne ha memoria. Allo stesso modo la cancellazione o correzione dei dati non sarebbe possibile. Tuttavia analizzando la blockchain dal punto di vista "security by design" si scopre che la pseudonimizzazione (disaccoppiamento fra i dati e l'identità dell'interessato) è garantito dal sistema di chiave asimmetrica adottato. In-

fatti non compare mai il nome del Data Owner ma solamente la sua chiave pubblica e l'integrità del dato è garantita dal suo hash. Per risolvere invece i problemi precedentemente citati si potrebbe adottare un approccio differente alle DLT non salvando affatto il dato all'interno del registro. La soluzione sarà affrontata più nel dettaglio nel capitolo 2.2.

1.3 File system in rete

Negli ultimi anni il cloud computing è diventato un settore predominante nel panorama informatico. L'esternalizzazione dei processi computazionali e dell'archiviazione è diventata importantissima per le aziende e i software in generale. La possibilità di utilizzare delle risorse esterne per effettuare attività di calcolo garantisce grossi risparmi e anche una potenza notevolmente superiore a quella che si potrebbe acquistare senza fare ricorso a questa tecnologia. L'archiviazione di dati in grossi data center consente inoltre di non dover gestire direttamente un carico di lavoro che soprattutto nelle piccole aziende sarebbe ingente. Il cloud computing si divide in tre tipologie principali:

- Infrastructure as a Service (IaaS): servizio che mette a disposizione API di alto livello per interagire con un livello fisico costituito da server, storage e connessioni di rete.
- Platform as a Service (PaaS): servizio che offre la possibilità di distribuire applicazioni nell'infrastruttura cloud. Non si ha il controllo dell'infrastruttura fisica ma solamente quello dell'applicativo.
- Software as a Service (SaaS): servizio che permette di eseguire gli applicativi direttamente nel cloud. Essi diventano accessibili da un interfaccia client come ad esempio un browser.

1.3.1 Soluzioni centralizzate

Attualmente leader di settore per i servizi cloud pubblici è Amazon Web Services. La quantità di data center sparsa per tutto il globo di cui dispone garantisce un'ottima copertura e conseguentemente delle ottime prestazioni. Gli strumenti messi a disposizione da AWS sono innumerevoli e spaziano dall'hosting di applicazioni web, gestione dei database, strumenti di machine learning, archiviazione e molto altro. Sebbene la presenza di numerosi server renda la rete AWS distribuita in questo caso viene definita centralizzata in quanto è sotto la gestione di un unica autorità. Amazon ha infatti il controllo totale dell'infrastruttura. La decisione di censurare o non supportare l'archiviazione di un determinato dato può essere intrapresa unilateralmente. Un discordo simile è valido anche per soluzioni come Google Drive. In questo caso però l'utenza a cui ci si rivolge è più consumer, cioè più rivolta alla creazione di uno spazio di archiviazione personale piuttosto che all'hosting di un sito o un grosso database.

1.3.2 IPFS

InterPlanetary File System [16] è un protocollo che permette di connettere dei dispositivi all'interno di una rete peer to peer. Lo scopo di IPFS è quello di archiviare e distribuire file, siti web, applicazioni e dati. Viene quindi creato un file system persistente e condiviso da tutti i nodi che non hanno bisogno di fidarsi l'uno dell'altro. Questo tipo di decentralizzazione permette di creare un internet più resiliente cioè resistente ad attacchi esterni e più difficile da censurare. Inoltre può velocizzare la fruizione di contenuti ottenendo i dati sempre dal nodo più vicino. Un oggetto IPFS è composto da:

- **Data**: contenuto in codice binario del file con dimensione minore di 256 kB.
- **Links**: un vettore di link ad altri oggetti IPFS. Ogni link è composto a sua volta dai campi *Name*, *Hash* (l'hash dei file linkato) e *Size* (dimensione cumulativa dell'oggetto linkato).

Come molti altri sistemi distribuiti IPFS utlizza i DAG (Grafi direzionati aciclici), nello specifico i Merkle DAG. In questa struttura ogni nodo ha come identificativo l'hash del suo contenuto. Quindi se il contenuto del file viene modificato ne consegue che anche il suo identificativo subirà delle variazioni e il nuovo file avrà un indirizzo diverso. Questo meccanismo chiamato *content addressing* non indica dove viene salvato il file quanto piuttosto un indirizzo basato sul contenuto del file. Inoltre i Merkle DAG sono immutabili e una variazione in un nodo (e quindi del suo identificativo) si ripercuote in tutti i discendenti, creando a tutti gli effetti un DAG nuovo e diverso. Conservando le vecchie versioni dei file IPFS riesce a tracciarne le variazioni nel tempo. A differenza di un Merkle Tree il DAG non richiede operazioni di bilanciamento e tutti i nodi possono ospitare del contenuto, non solamente le foglie.

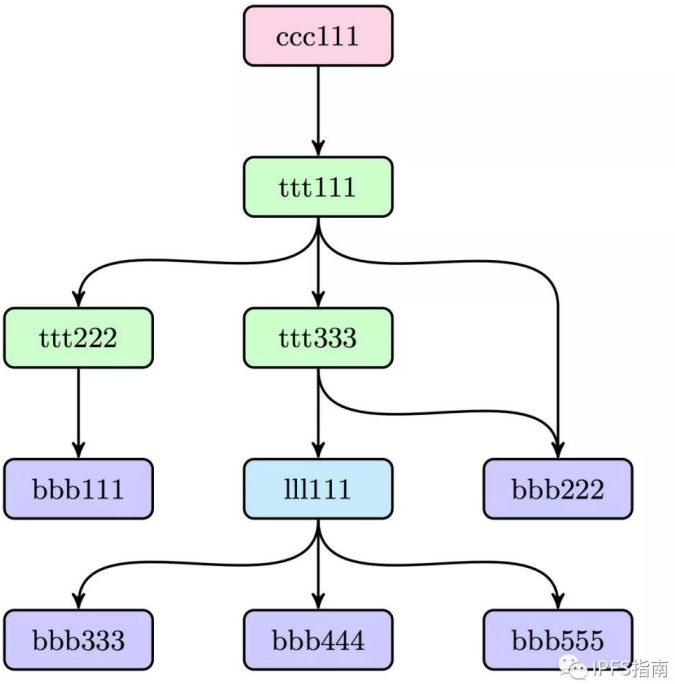

Figura 1.4: Un esempio di Merkle DAG [2]

Il processo di routing invece è attuato con l'aiuto di una *Distributed Hash Table* (DHT) mantenuta da tutti i nodi. Essa contiene i riferimenti agli

oggetti e consente di rintracciare i file all'interno della rete. Quando un nodo vuole cercare un file interroga i nodi vicini che controlleranno la loro porzione di hash table (si tratta di un semplice confronto chiave-valore). In caso di successo viene restituito il valore cercato, altrimenti la richiesta viene rimbalzata ad altri nodi meno vicini.

Capitolo 2

Data Marketplace

I Data Marketplace costituiscono una valida alternativa alla gestione centralizzata dei dati. Il primo fu lanciato nel 2017 dalla IOTA Foundation come Proof of Concept [17]. Evolvendosi sono diventati un mercato digitale per i dati degli utenti, che possono essere comprati proprio come degli articoli qualsiasi. L'obiettivo è quello di rispettare le regole e condizioni dettate dalla legge (con uno speciale riferimento alle direttive GDPR). All'interno di queste piattaforme i Data Provider sono coloro che rendono disponibili i propri dati mentre i Data Consumer sono coloro ne fanno utilizzo. L'incentivo all'utilizzo per i provider è di tipo monetario perché ad ogni transazione corrisponde un ammontare di una certa criptovaluta, ricompensando quindi tutti coloro che mettono a disposizione i loro dati. I dati presenti nel marketplace possono essere generati da qualsiasi dispositivo, sia esso uno smartphone, smartwatch oppure sensori di vario tipo come quelli presenti nelle più recenti automobili smart. La decentralizzazione è una prerogativa importante per queste piattaforme, infatti per rendere consapevole l'utente e trasparenti le transazioni è opportuno che i Marketplace si appoggino alla blockchain. In questo modo la tracciabilità del dato e l'immutabilità sono assicurate in principio e la privacy può essere ottenuta attraverso meccanismi di controllo d'accesso alle risorse. Esiste inoltre la possibilità che questa architettura si appoggi ad una blockchain privata, fornendo quindi ulteriori garanzie circa la provenienza dei

potenziali acquirenti. In questo caso però i Data Owner dovrebbero essere sottoposti ad una fase iniziale di controllo circa la loro identità reale prima di poter essere inseriti nella blockchain.

2.1 Scenari reali

Gli scenari per l'utilizzo di questa tecnologia sono moltissimi e si coniugano particolarmente bene con la mobilità smart. Poniamo ad esempio che Bob, grazie al suo smartphone, salvi i propri dati di latitudine e longitudine ogni minuto mentre si trova in auto. Successivamente una società di car sharing interessata al dato come ad esempio BlaBlaCar potrà richiedere a Bob l'accesso alle sue informazioni per effettuare le sue analisi e ne otterrà l'accesso attraverso un micropagamento. Sempre in ambito mobilità smart si può immaginare uno scenario in cui i passeggeri di un autobus utilizzino un'applicazione che una volta connessa al Wi-Fi del veicolo comunichi in tempo reale la posizione, numero di passeggeri a bordo, temperatura e molte altre informazioni a tutti gli altri utenti dell'applicazione che invece stanno aspettando l'autobus. Si potrebbe inoltre monitorare in tempo reale le prestazioni di ogni tipo di veicolo grazie alla diagnostica di bordo. Connettendo infatti un lettore OBD ad una architettura simile si possono comunicare i dati raccolti in tempo reale ad un officina meccanica che potrà poi analizzare. Per quanto concerne invece la gestione dei dati personali si può pensare ad applicativi come Digi.me [18] che si appoggino però alla blockchain, in modo da rendere ancora più trasparente e sicura la condivisione delle informazioni. In questo caso Bob disporrebbe di un applicazione per smartphone che raccoglie tutte le sue attività sui social e avrebbe la possibilità di consultarle e selezionare ciò che vuole vendere alle aziende interessate. Anche il settore IoT è coinvolto nella creazione di grossissime quantità di dati, si pensi ad esempio alla domotica che richiede il salvataggio dei log di tutti i dispositivi connessi. Le aziende sono interessate ad accedere a questi registri per capire l'utilizzo che viene effettuato degli apparecchi e per compiere analisi di

mercato. La condivisione di queste informazioni può avvenire in modo sicuro e trasparente utilizzando il Marketplace [19]. I benefici di cui godrebbe il settore IoT sono molteplici, sia di natura tecnica perché le aziende produttrici non dovrebbero preoccuparsi di costruire un'infrastruttura backend per il salvataggio dei dati, che di natura economica in quanto si potrebbero sperimentare nuovi modelli di businness. Infatti i prodotti IoT potrebbero addirittura essere forniti gratuitamente in cambio della condivisione dei dati d'utilizzo con il produttore [20].

2.2 Architettura

L'applicazione si appoggia ovviamente ad una piattaforma blockchain che nel caso in esame è Ethereum. Di conseguenza utilizza la sua criptovaluta Ether per poter effettuare le transazioni. Il costo dell'operazione viene espresso come gas e non è altro che una piccola tassa da pagare indirizzata al miner che validerà la transazione. Infatti in Ethereum (come nella maggior parte delle blockchain) si può leggere il registro gratuitamente ma bisogna pagare quando si vuole scrivere. In conformità con il diritto all'oblio del GDPR non viene caricato il dato direttamente sul registro ma viene salvato solamente un riferimento ad esso. L'informazione vera e propria è infatti cifrata e salvata su un'infrastruttura cloud come Google Drive e Amazon Web Services oppure in un ambiente distribuito come IPFS. Il riferimento potrebbe essere il digest del dato oppure un vero e proprio indirizzo a cui accedere per consultare l'informazione. Vari servizi di file system in rete accorrono in aiuto e rendono disponibile la condivisione dei file tramite url. Lo Smart Contract regola quindi il rilascio di questo url presente nella blockchain, che sarà disponibile solo a pagamento effettuato. Per quanto riguarda la cancellazione e modifica dei dati, non essendo possibile la manipolazione del link a livello di catena di blocchi deve essere gestita direttamente nel luogo in cui è stato memorizzato. Quindi se il dato è errato può essere modificato mantenendo lo stesso url. Se invece è richiesta la cancellazione eliminando il dato l'url

punterà ad un dato inesistente. È naturale però che se qualcuno dei nodi ha avuto accesso in precedenza all'informazione nulla può garantire la rimozione anche da parte loro. In questo caso specifico si è scelto di caricare il file su Google Drive e sfruttare la condivisione tramite link come riferimento al file. Se il proprietario volesse perciò rimuovere il dato dovrebbe solamente eliminarlo dal Cloud, rendendo il riferimento inutile perché punterebbe ad un indirizzo inesistente. Le due parti interagiscono grazie ad un particolare protocollo chiamato Smart Contract che rende disponibile il dato nel sito web dopo aver ricevuto il pagamento necessario.

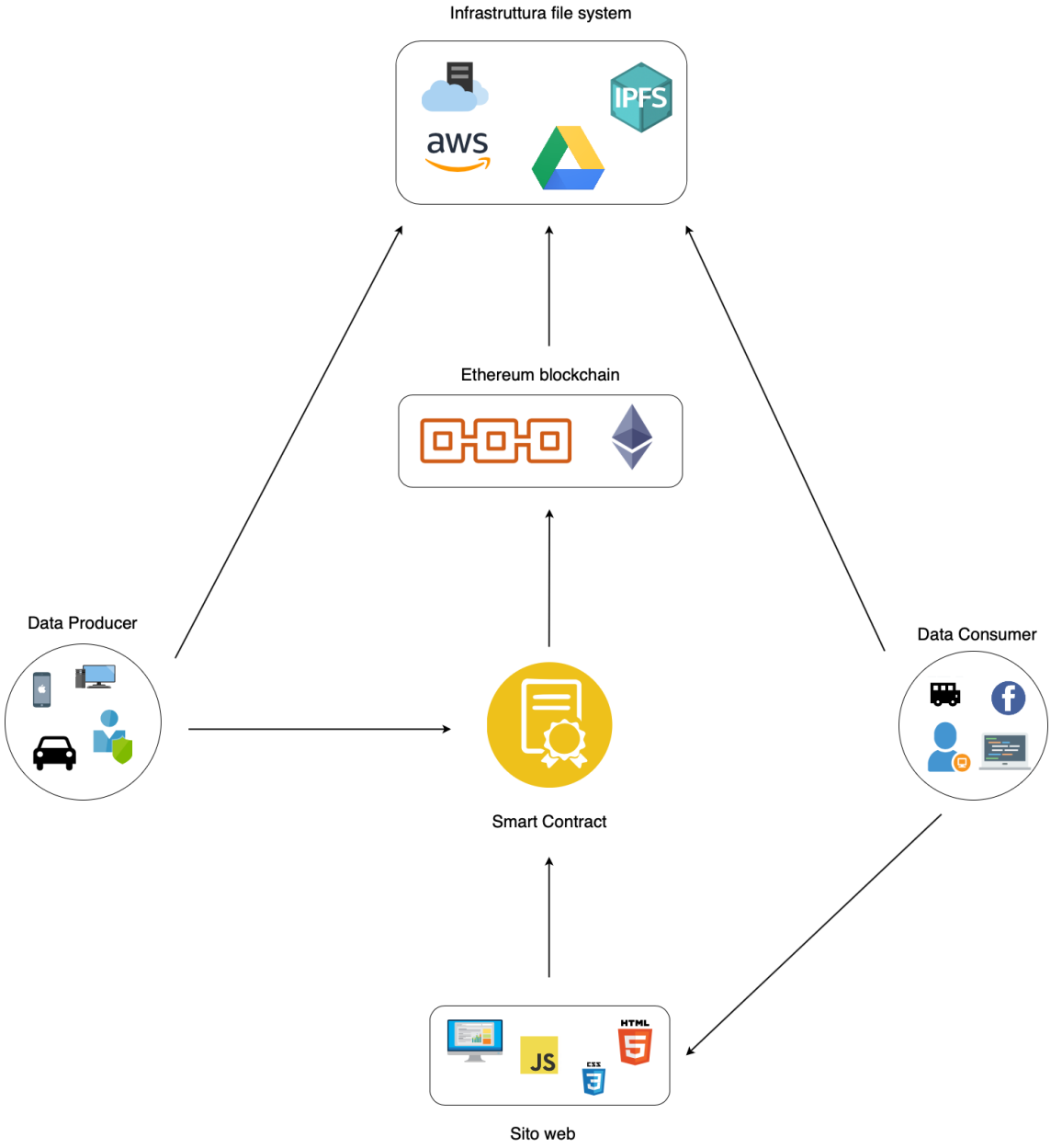

Figura 2.1: Struttura di un Data Marketplace

2.3 Data Shop

Ispirandomi quindi al Pet Shop di Truffle [21] ho deciso di realizzare una versione dell'applicativo che supportasse l'acquisto di dati. All'interno di questa pagina web è possibile visualizzare un insieme di dati da acquistare. Nel momento in cui si clicca sul bottone "Buy" verrà reso disponibile il link a Google Drive contenente il dato[22].

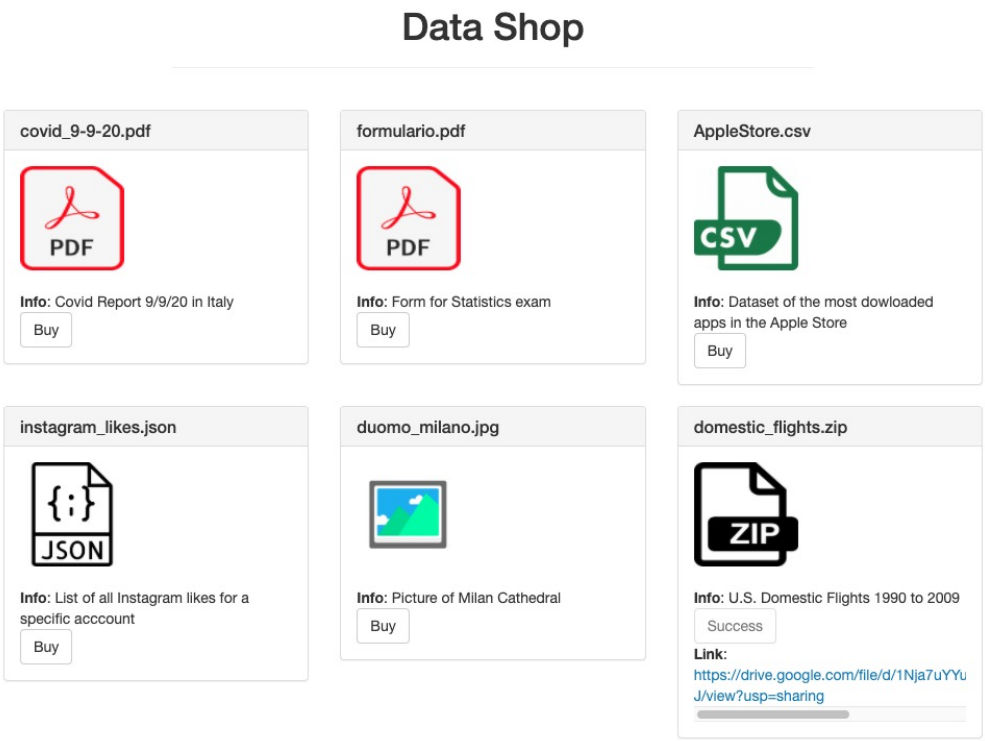

Figura 2.2: Pagina di acquisto dei dati

2.3.1 Smart Contract

Per poter gestire l'ottenimento del dato è necessario utilizzare una speciale struttura capace di eseguire del codice che stipula un accordo fra due o più parti: lo Smart Contract. Questo speciale contratto è scritto nel linguaggio Solidity e viene eseguito all'interno della piattaforma Ethereum. L'attivazio-

ne dello stesso avviene grazie ad una specifica transazione verso un nodo e al gas che gli viene fornito. Nello specifico lo Smart Contract del Data Shop mantiene una semplice struttura dati con gli indirizzi di coloro che acquistano i dati nella piattaforma. Questa lista chiamata `buyers` può essere anche restituita con l'apposito metodo, mentre il valore 100 è solamente arbitrario e costituisce il massimo numero di dati che possono essere presenti. Dopo l'acquisto l'indirizzo dell'utente viene infatti inserito in un vettore che tiene traccia di coloro che hanno accesso ai file. Di seguito l'implementazione di quanto appena descritto:

```
pragma solidity ^0.5.0;

contract Purchase {

    address[100] public buyers;

    // Buying a data
    function buy(uint dataId) public returns (uint) {
        require(dataId >= 0 && dataId <= 100);
        buyers[dataId] = msg.sender;
        return dataId;
    }

    // Retrieving the buyers
    function getBuyers() public view returns (address[100]
       memory) {
        return buyers;
    }
}
```

Il metodo `buy(uint dataId)` dopo aver preso in input l'identificativo univoco del dato da acquistare ed aver verificato che è contenuto all'interno del range specificato assegna in posizione `dataId` del vettore `buyers` l'indirizzo dell'acquirente. Il secondo metodo `getBuyers()` invece si occupa unicamente di restituire in output la lista `buyers`.

2.3.2 Test

Lo sviluppo è stato agevolato dall'utilizzo del framework Truffle [23]. Grazie a questo pacchetto è possibile scrivere applicazioni decentralizzate su Ethereum, compilando gli Smart Contract ed effettuando la migrazione degli stessi sulla blockchain. Truffle inoltre permette di effettuare test degli Smart Contract sviluppati:

```
pragma solidity ^0.5.0;

import "truffle/Assert.sol";
import "truffle/DeployedAddresses.sol";
import "../contracts/Purchase.sol";

contract TestPurchase {
    // The address of the purchase contract to be tested
    Purchase purchase = Purchase(DeployedAddresses.Purchase()
        );

    // The id of the data that will be used for testing
    uint expectedDataId = 3;

    // The expected owner of bought data is this contract
    address expectedBuyer = address(this);

    // Testing the buy() function
    function testUserCanBuyData() public {
        uint returnedId = purchase.buy(expectedDataId);
        Assert.equal(returnedId, expectedDataId, "Purchase of
            the expected data should match what is returned."
           );
    }

    // Testing retrieval of a single data's owner
    function testGetBuyerAddressByDataId() public {
        address buyer = purchase.buyers(expectedDataId);
        Assert.equal(buyer, expectedBuyer, "Owner of the
           expected data should be this contract");
```

```
    }

    // Testing retrieval of all data owners
    function testGetBuyerAddressByDataIdInArray() public {
        // Store buyers in memory rather than contract's
           storage
        address[100] memory buyers = purchase.getBuyers();
        Assert.equal(buyers[expectedDataId], expectedBuyer, "
           Owner of the expected data should be this contract
           ");
    }

}
```

I test sono utili per verificare il corretto funzionamento dello Smart Contract. Dopo aver importato i contratti necessari vengono definite tre variabili: `purchase`, `expectedDataID` e `expectedBuyer`. Contengono rispettivamente l'istanza del contratto di acquisto migrato nella blockchain, l'identificativo del dato usato per i test e l'indirizzo corrente. Di seguito sono presenti tre metodi, il primo dei quali `testGetBuyerAddressByData()` simula l'acquisto del dato definito in precedenza e confronta l'identificativo ritornato con quello atteso. Il secondo metodo `testGetBuyerAddressByDataID()` confronta invece l'identificativo dell'utente con quello associato al compratore del dato e verifica che i due corrispondano. La terza ed ultima funziona `testGetBuyerAddressByDataIdInArray()` si accerta che l'indirizzo del compratore sia uguale a quello presente nel vettore `buyers` in corrispondenza del dato acquistato.

2.4 Strumenti utilizzati

L'applicativo si appoggia Ganache [24], una blockchain personale e locale per lo sviluppo di applicazioni distribuite. Essa permette di testare agevolmente gli Smart Contract ed emula le funzioni del network Ethereum, includendo account multipli e un certo numero di Ether.

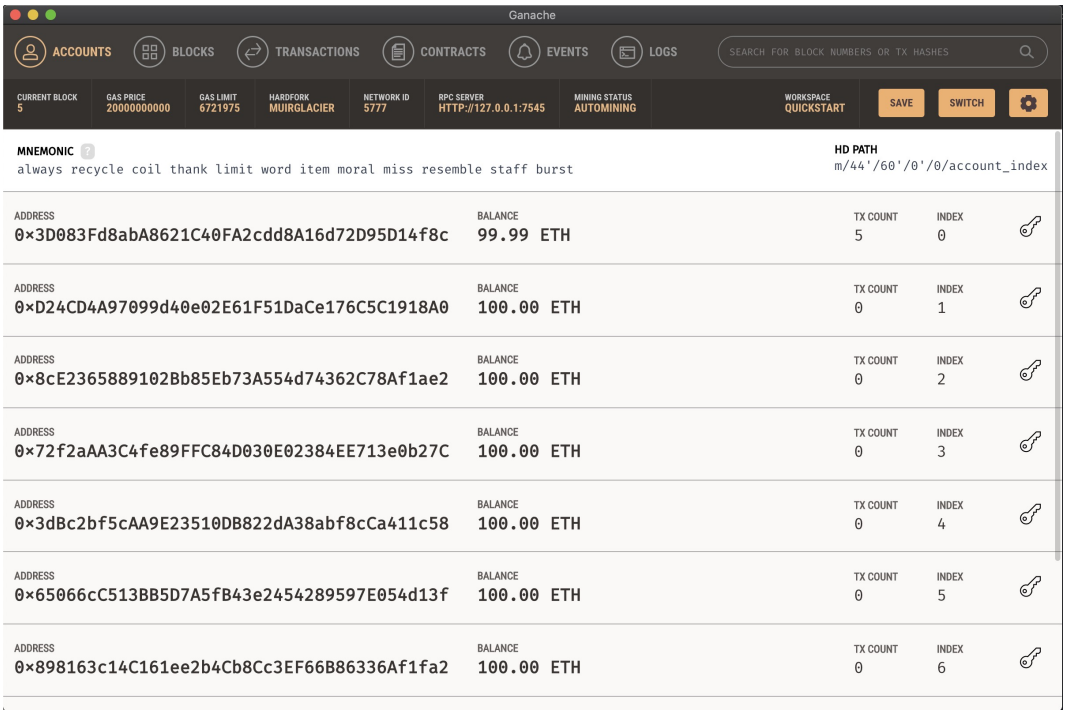

Figura 2.3: GUI di Ganache

Per poter interagire attraverso il browser viene utilizzata l'API Javascript per Etherium web3.js [25] e l'estensione MetaMask [26] per Google Chrome e Mozilla Firefox. Collegandosi all'indirizzo 127.0.0.1:7545 (la porta riservata a Ganache) permette di importare l'account grazie alle 12 parole segrete che costituiscono il "Seed". Ad ogni acquisto di un dato si apre un dialog riepilogativo con ammontare di gas da spendere per confermare le transazioni.

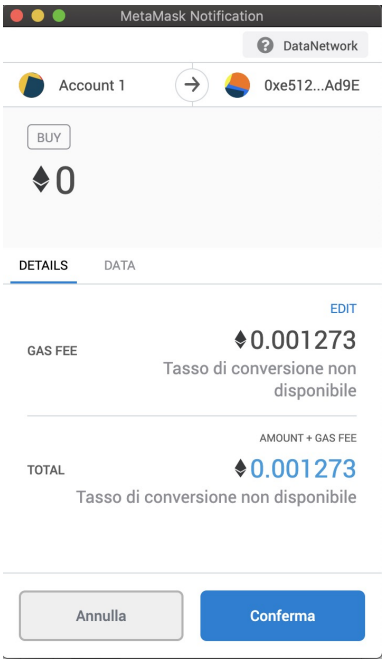

Figura 2.4: Conferma transazione con MetaMask

Capitolo 3

Scenario sperimentale

Come abbiamo visto nel paragrafo 1.2.2 il salvataggio di dati direttamente sulla blockchain può risultare problematico alla luce del nuovo GDPR. In particolare si è discusso di come la blockchain sia per sua natura incompatibile con il diritto all'oblio *("right to be forgotten")*. Cancellare i dati non è possibile in un sistema progettato per essere immutabile. La soluzione è quindi quella di memorizzare altrove i dati e di inserire un riferimento nella catena. Tuttavia tenendo a mente scenari nei quali vengono prodotte grosse quantità di dati in poco tempo i file system utilizzati a supporto della blockchain devono garantire delle latenze accettabili. Tempi di caricamento alti aggraverebbero ancor di più il processo di salvataggio. In questo capitolo quindi si è voluto approfondire il comportamento di diverse piattaforme nel momento in cui vengono stressate con il caricamento di dati.

3.1 Il dataset

Il dataset utilizzato raccoglie il tragitto di diversi bus di Rio de Janeiro [27]. In particolare queste tracce di percorsi hanno una posizione aggiornata ogni minuto. Sono stati poi simulati un certo numero di passeggeri dei bus, da 10 a 100, che ogni minuto effettuano l'upload di dati che possono rap-

presentare quelli raccolti da sensori quindi temperatura, posizione o altro. Nell'esperimento i dati sono di due tipi:

- Dati di piccole dimensioni: file json di 100 byte circa che simulano l'invio di piccole quantità di informazioni.
- Dati di medie dimensioni: immagini di circa 1 MB che simulano l'invio di quantità maggiori di informazioni.

Figura 3.1: Tragitto di un'ora di 10 bus [3]

Le piattaforme utilizzate per il testing sono tre: IPFS, Google Drive e Amazon Web Services .

3.2 Struttura dei test

Per poter automatizzare l'upload dei dati è stato utilizzato uno script node.js [28] opportunamente modificato per supportare le API dei diversi

servizi utilizzati.

3.2.1 Script IPFS

L'upload dei file verso il network IPFS avviene utilizzando l'API Infura. La funzione che si occupa del caricamento è la seguente:

```
const publish = async (b, id, json, prop) => {
  console.log;
  const ipfs = !prop ? ipfsProp : ipfsService;
  let startTS = -1,
    finishTS = -1;
  try {
    //Start operations
    startTS = new Date().getTime();
    const raceRes = await Promise.race([
      //first
      new Promise(async (resolve, reject) => {
        for await (const result of ipfs.add(JSON.stringify(
            json))) {
          //console.log(result);
        }
        resolve(true);
      }),
      //second
      sleep(timeoutValue),
    ]);
    if (raceRes) {
      finishTS = new Date().getTime();
      // Latency measures
      r = finishTS - startTS;
      // Log result
      console.log(prop + ') bus ' + b + ': ' + r + 'ms');
      fs.appendFile(
        bus[b].csv[prop],
        startTS + ',' + finishTS + ',' + id + '\n',
        (err) => {
          if (err) throw err;
```

```
        }
      );
    } else {
      throw new Error('ipfs add timeout');
    }
  } catch (err) {
    console.log(prop + ')' + b + ': ' + err);
    fs.appendFile(
      bus[b].csv[prop],
      startTS + ',' + finishTS + ',' + id + '\n',
      (err) => {
        if (err) throw err;
      }
    );
  }
};
```

Dopo aver verificato il valore della costante `ipfs` e aver inizializzato i timestamp viene eseguita la promessa JavaScript per il caricamento del contenuto richiamando il metodo `ipfs.add`. Successivamente viene valutato lo stato della promessa e se terminata vengono salvati i timestamp di inizio e fine del'upload su di un file csv per poter analizzare i dati.

3.2.2 Script Google Drive

La seconda piattaforma su cui sono stati condotti i test di performance è Google Drive. L'API utilizzata in questa occasione è quella nativa per node.js versione 3. Dopo aver ottenuto l'autorizzazione dell'account nel quale caricare i contenuti la funzione `publish` è stata adattata nel seguente modo:

```
const publish = async (b, id, json, prop, auth) => {
  const drive = google.drive({ version: 'v3', auth });
  let startTS = -1,
    finishTS = -1;
  data = JSON.stringify(json, null, 2);

  try {
```

```
//Start operations
startTS = new Date().getTime();
if (image) {
  var fileName = 's' + startTS + '.jpg';
  var fileMetadata = {
    name: fileName,
    parents: [global.folderId],
  };
  var media = {
    mimeType: 'image/jpeg',
    body: fs.createReadStream(imagePath),
  };
} else {
  var fileName = 's' + startTS + '.json';
  var fileMetadata = {
    name: fileName,
    parents: [global.folderId],
  };
  var media = {
    mimeType: 'application/json',
    body: data,
  };
  fs.writeFileSync(fileName, data);
}
//first
limiter.schedule(
  () =>
    new Promise(async (resolve, reject) => {
      drive.files.create(
        {
          resource: fileMetadata,
          media: media,
          fields: 'id',
        },
        function (err, res) {
          if (err) {
            // Handle error
```

```
                console.log('Error uploading do gdrive: ' +
                    err);
                !image ? fs.unlinkSync(fileName) : '';
              } else {
                // if succeded
                finishTS = new Date().getTime();
                // Latency measures
                r = finishTS - startTS;
                // Log result
                console.log(prop + ') bus ' + b + ': ' + r +
                    'ms');
                fs.appendFile(
                  bus[b].csv,
                  startTS + ',' + finishTS + ',' + id + '\n',
                  (err) => {
                    if (err) throw err;
                  }
                );
                !image ? fs.unlinkSync(fileName) : '';
              }
            }
          );
          resolve(true);
        })
    );
    //second
    sleep(timeoutValue);
  } catch (err) {
    console.log(prop + ')' + b + ': ' + err);
    fs.appendFile(
      bus[b].csv,
      startTS + ',' + finishTS + ',' + id + '\n',
      (err) => {
        if (err) throw err;
      }
    );
  }
};
```

In questo script viene esplicitamente distinto il caso in cui si tratti di un immagine o di un file json. Questo perché Google Drive necessita che sia specificato un `mimeType` prima di effettuare il caricamento. Le immagini vengono caricate direttamente recuperandole dal percorso nel file system, mentre per i dati sulla posizione viene prima creato un file json locale e successivamente caricato. Come nel caso di IPFS vengono salvati sottoforma di file csv i timestamp di caricamento dei dati. Il nuovo parametro da impostare è `folder.id`, l'identificativo della cartella in cui si vuole caricare il file. Si può inoltre notare l'utilizzo dell'oggetto `limiter` per non sforare il limite di richieste consentito dall'API. Infatti utilizzando questo oggetto del modulo `Bottleneck` le richieste vengono inviate ad intervalli di 110 millisecondi.

3.2.3 Script AWS

L'ultima piattaforma utilizzata è Amazon Web Services. In particolare il caricamento viene effettuato su di un Bucket S3. Anche in questo caso la funzione modificata è publish:

```
const publish = async (b, id, json, prop) => {
  let startTS = -1,
    finishTS = -1;
  data = JSON.stringify(json, null, 2);

  try {
    //Start operations
    startTS = new Date().getTime();
    if (image) {
      var fileName = 's' + startTS + '.jpg';
      data = fs.createReadStream(imagePath);
    } else {
      var fileName = 's' + startTS + '.json';
    }

    // Setting up S3 upload parameters
    const params = {
      Bucket: BUCKET_NAME,
```

```
    Key: fileName, // File name you want to save as in S3
    Body: data
};
  //first
  limiter.schedule(
    () =>
    new Promise(async (resolve, reject) => {
      // Uploading files to the bucket
      s3.upload(params, function (err, data) {
        if (err) {
          console.log('Error uploading do AWS: ' + err);
          throw err;
        } else {
          // if succeded
          finishTS = new Date().getTime();
          // Latency measures
          r = finishTS - startTS;
          // Log result
          console.log(prop + ') bus ' + b + ': ' + r + 'ms'
              );
          fs.appendFile(
            bus[b].csv,
            startTS + ',' + finishTS + ',' + id + '\n',
            (err) => {
              if (err) throw err;
            }
          );
        }
      });
      resolve(true);
    })
  );
  //second
  sleep(timeoutValue);
} catch (err) {
  console.log(prop + ')' + b + ': ' + err);
  fs.appendFile(
    bus[b].csv,
```

```
      startTS + ',' + finishTS + ',' + id + '\n',
      (err) => {
        if (err) throw err;
      }
    );
  }
};
```

Anche in questa circostanza viene utilizzato l'oggetto `limiter` per dilazionare le richieste in intervalli regolari. Non c'è più l'identificativo della cartella, sostituito dal nome del Bucket contenente i file.

3.3 Risultati

Come spiegato in precedenza i test hanno come obiettivo quello di misurare le prestazioni delle tre diverse soluzioni adottate in diversi scenari. Le variabili sono costituite sia dal tipo di file caricato (file testuale o immagine) ma anche dalla congestione della rete. Il carico di lavoro aggiuntivo viene simulato aumentando il numero di passeggeri presenti nel bus. Il numero minimo è 10 mentre il massimo è 100. Così facendo si vuole individuare quale piattaforma offre i tempi di caricamento migliori e quale supporta meglio un grosso numero di utenti, senza dimenticare il fattore legato agli errori che si possono generare. Una simulazione ha la durata di 15 minuti circa ed invia esattamente 15 messaggi per ogni utente. Di seguito quindi vengono riportati i grafici relativi alle latenze misurate. Le colonne blu si riferiscono ai file testuali di piccole dimensioni mentre quelle rosse alle immagini di medie dimensioni.

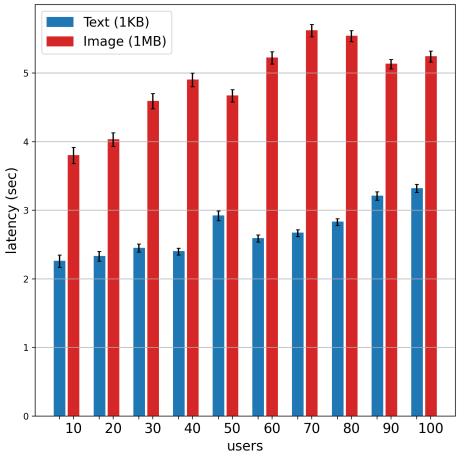

Figura 3.2: IPFS

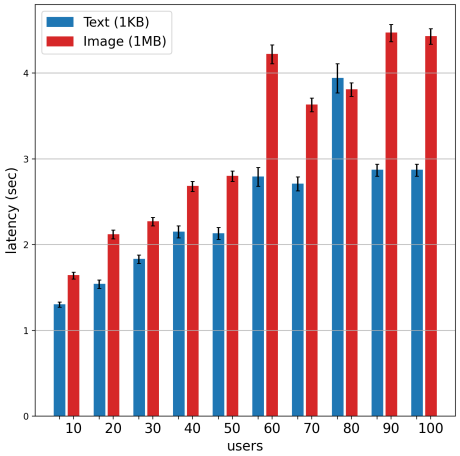

Figura 3.3: Google Drive

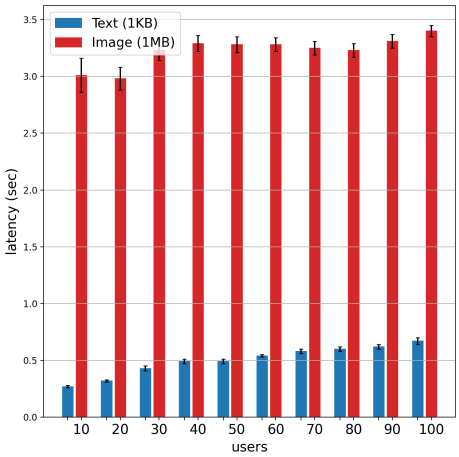

Figura 3.4: Amazon Web Services

Tutte le rilevazioni sono state effettuate a parità di connessione ad Internet in modo da essere confrontabili fra loro. L'intervallo di confidenza è stato impostato al 95% in modo da eliminare i valori outlier troppo bassi e troppo alti che potrebbero influenzare i grafici. Si può notare che l'aumento rilevato nei tempi per la tecnologia IPFS è simile sia per quanto riguarda i file testuali che le immagini. Infatti con l'aumentare del numero di utenti cresce anche la latenza ma in modo omogeneo. Lo stesso si può dire anche nel

caso di AWS ma non di Google Drive. Le prestazioni di quest'ultimo infatti peggiorano notevolmente con l'aumentare del numero di utenti, evidenziando qualche problema nel supportare un carico elevato di richieste. Per quanto riguarda le prestazioni in relazione del tipo di contenuto AWS è assolutamente la soluzione migliore nel caricamento di file di piccole dimensioni. Infatti le latenze registrate sono 3/4 volte inferiori a quelle delle altre due piattaforme. Altro pregio di AWS è la stabilità perché le latenze sono poco sensibili al numero di utenti, quasi costanti. Nel caricamento di file di medie dimensioni si può notare che per un numero ristretto di utenti Google Drive risulta essere la piattaforma più performante, salvo poi peggiorare a partire da 60 utenti collegati in poi. La tecnologia preferibile sembrerebbe essere quindi quella offerta da Amazon, sia in termini di performance assoluta che di scalabilità del sistema. È importante sottolineare la diversa natura dei servizi, in quanto IPFS è un protocollo aperto e accessibile tutti gratuitamente mentre gli altri due possono essere a pagamento. Google Drive offre infatti un servizio gratuito con 15 GB di spazio di archiviazione e uno a pagamento chiamato Google One che potrebbe avere performance migliori. Amazon Web Services invece è gratuito fino a 2000 richieste (PUT, COPY, POST o LIST) e poi costa 0,0053$ ogni 1000 richieste.

3.3.1 Risultati con connessione differente

I risultati precedenti sono stati ottenuti effettuando i test con una connessione che garantisce circa 100 Mbps in upload. Tuttavia c'è stata la possibilità di svolgere i test su IPFS e Google Drive anche su una connessione differente. Quest'ultima offre una velocità inferiore, circa 20 Mbps in upload, ma utilizzandola si è riscontrato avere una variazione di velocità inferiore nel corso del tempo.

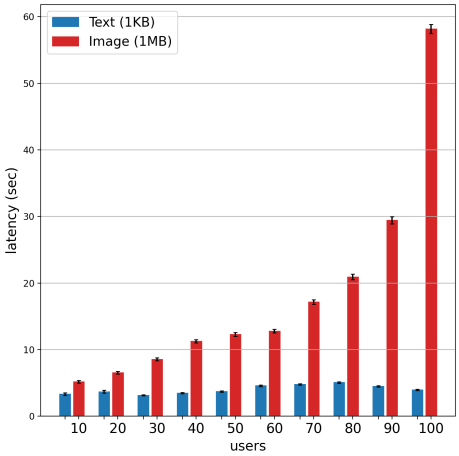

Figura 3.5: IPFS con connessione stabile

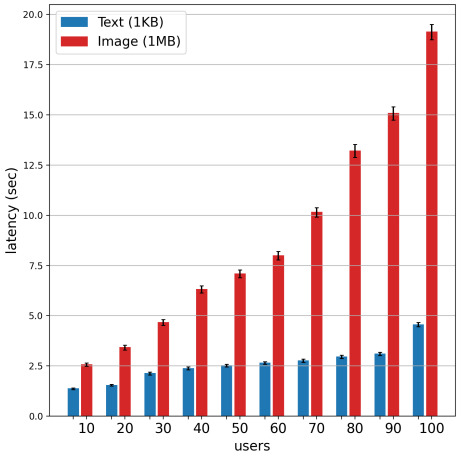

Figura 3.6: Google Drive con connessione stabile

Con questa connessione più lenta ma stabile è più semplice osservare l'evoluzione delle latenze al variare del carico di lavoro richiesto. Nonostante latenze registrate siano ovviamente più alte in valore assoluto è ben evidenziato il comportamento delle due piattaforme. Infatti entrambe mal supportano l'aumento del carico di lavoro. In particolare con le immagini su IPFS e 100 utenti collegati i valori registrati esplodono, probabilmente a causa del fatto che il nodo non riesce a distribuire adeguatamente le richieste all'interno della rete. Per quanto riguarda i file di testo invece IPFS presenta un andamento costante. Utilizzando questa rete il comportamento caratteristico di Google Drive è ben visibile, infatti presenta una crescita esponenziale delle latenze. Probabilmente è dovuto alla natura del servizio, pensato come spazio di archiviazione personale piuttosto che come infrastruttura per la distribuzione rapida e affidabile di contenuti.

Capitolo 4

Conclusioni

Nel corso del presente elaborato si è introdotto il mondo dei registri decentralizzati. Queste tecnologie, non essendo più ormai l'ultima novità nel campo informatico, sono chiamate alla prova della maturità. Avendo consolidato la posizione e utilità nel settore delle criptovalute si stanno sperimentando innumerevoli nuove applicazioni, alla ricerca di quelle che possono più beneficiare di questa piattaforma. Descrivendo il funzionamento e i principali protagonisti si è avuto modo di capire quali siano i concetti alla base del funzionamento della blockchain. Inoltre evidenziandone i pregi quali immutabilità, sicurezza e tracciabilità si è compreso per quale motivo l'utilizzo delle DLT in generale può costituire un salto di qualità per diversi settori. Tuttavia abbiamo visto che esiste ancora qualche problema che ne inficia l'adozione su larga scala. In particolare la scalabilità impedisce di concorrere con network più consolidati ed efficienti come i principali circuiti finanziari. Esistono delle soluzioni nuove che promettono di risolvere il problema quali IOTA e RadixDLT. Successivamente ci si è concentrati sull'utilizzo della blockchain per la gestione dei dati personali con focus dedicato al GDPR. Questo approccio richiede l'ausilio di servizi di archiviazione dati online. Sono state quindi presentate tre alternative quali Amazon Web Services, Google Drive e IPFS. Nella parte centrale della tesi si è presentata nello specifico una possibile architettura per un servizio di Data Marketplace basato su Ethereum.

Grazie ad uno Smart Contract è possibile realizzare un sistema decentralizzato in cui gli utenti condividono in modo sicuro i propri dati personali e monetizzano a partire dalla loro condivisione. Nell'ultimo capitolo infine vengono analizzate le performance dei vari servizi cloud citati per capire in che misura influiscono sul corretto funzionamento della blockchain. È emerso che Google Drive e IPFS non supportano al meglio un grosso carico di lavoro, mentre negli stress test Amazon Web Services è sembrato essere più rapido e consistente al variare del numero di utenti.

4.0.1 Problemi riscontrati

L'idea iniziale per il Data Shop era di realizzarlo utilizzando la piattaforma RadixDLT. Quest'ultima infatti è molto promettente ma non offre degli strumenti adeguati per gli sviluppatori e quindi dopo numerosi tentativi ci si è orientati verso piattaforme più solide e mature come Ethereum. Questo va a sottolineare come nuove tecnologie di questo tipo abbiano bisogno di tempo sia per proporre soluzioni tecniche che per raggiungere una platea di utenti tale da creare una community di supporto.

4.0.2 Sviluppi futuri

Il Data Shop presentato potrebbe essere integrato con una interfaccia lato client per il caricamento dei dati e l'inserimento automatico del riferimento nella blockchain. Inoltre un altro aspetto interessante da ampliare potrebbe essere la condivisione dei dati con entità terze rispetto al Data Consumer, sollevando nuove soluzioni tecniche e regolamentari.

Bibliografia

[1] Report di MarketsandMarkets. *https://www.marketsandmarkets.com/Market-Reports/blockchain-technology-market-90100890.html*.

[2] Merkle DAG. *https://www.programmersought.com/article/63094327951/*.

[3] Mirko Zichichi, Stefano Ferretti, and Gabriele D'Angelo. On the efficiency of decentralized file storage for personal information management systems. In *Proc. of the 2nd International Workshop on Social (Media) Sensing, co-located with 25th IEEE Symposium on Computers and Communications 2020 (ISCC2020)*, pages 1–6. IEEE, 2020.

[4] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[5] Bitcoin Consumption. *https://digiconomist.net/bitcoin-energy-consumption*.

[6] Vitalik Buterin. Ethereum White Paper: A Next Generation Smart Contract Decentralized Application Platform. 2013.

[7] Ittay Croman, Kyle; Eyal. On Scaling Decentralized Blockchains. 2016.

[8] Ethereum GitHub Wiki. 2019.

[9] Serguei Popov. The Tangle. 2018.

[10] Dan Hughes. Why DAGs Don't Scale Without Centralization. 2018.

[11] Scaling dlt to over 1m tps on google cloud. *https://www.radixdlt.com/post/scaling-dlt-to-over-1m-tps-on-google-cloud/*.

[12] Josh Primero Stephen J. Thornton Florian Cäsar, Daniel P. Hughes. Cerberus A Parallelized BFT Consensus Protocol for Radix Centralization. 2020.

[13] Cambrige Analytica. *https://www.theguardian.com/news/2018/may/06/cambridge-analytica-how-turn-clicks-into-votes-christopher-wylie*.

[14] Google Nightingale. *https://www.theatlantic.com/technology/archive/2019/11/google-project-nightingale-all-your-health-data/601999/*.

[15] Regolamento (UE) 2016/679 del Parlamento europeo e del Consiglio, del 27 aprile 2016, relativo alla protezione delle persone fisiche con riguardo al trattamento dei dati personali, nonché alla libera circolazione di tali dati e che abroga la direttiva 95/46/CE (regolamento generale sulla protezione dei dati). 2016.

[16] IPFS. *https://ipfs.io*.

[17] IOTA Marketplace. *https://data.iota.org/*.

[18] Digi.me. *https://digi.me*.

[19] Ji-Sun Park, Taek-Young Youn, Hye-Bin Kim, Kyung-Hyune Rhee, and Sang-Uk Shin. Smart Contract-Based Review System for an IoT Data Marketplace. *Sensors*, 18(10):3577, 2018.

[20] Kazim Rifat Özyilmaz, Mehmet Doğan, and Arda Yurdakul. IDMoB: IoT Data Marketplace on Blockchain. In *2018 crypto valley conference on blockchain technology (CVCBT)*, pages 11–19. IEEE, 2018.

[21] Ethereum Pet Shop. *https://www.trufflesuite.com/tutorials/pet-shop*.

[22] Data Shop GitHub repository. *https://github.com/riccardocavallin/data-shop*.

[23] Truffle. *https://github.com/trufflesuite/truffle*.

[24] Ganache. *https://github.com/trufflesuite/ganache*.

[25] web3.js. *https://github.com/ethereum/web3.js/*.

[26] MetaMask. *https://metamask.io*.

[27] Daniel Dias and Luís Henrique Maciel Kosmalski Costa. CRAWDAD dataset coppe-ufrj/riobuses (v. 2018-03-19). Downloaded from https://crawdad.org/coppe-ufrj/RioBuses/20180319/RioBuses, March 2018. traceset: RioBuses.

[28] GitHub repository per i test sulle perfomance dei file system in rete. *https://github.com/riccardocavallin/testingGDrive*.