Architettura modulare e Crittografia di Solana

Martin Tomassi

19 maggio 2025

Indice

Capitolo 1

Introduzione a Solana

Solana nasce con l'obiettivo di superare i limiti delle generazioni precedenti, in termini di scalabilità, velocità di esecuzione e costi di transazione. Il crescente utilizzo delle principali blockchain, come Bitcoin ed Ethereum, ha evidenziato alcuni problemi strutturali: tempi di validazione elevati, commissioni in crescita e limitazioni nel numero di transazioni al secondo. Questi vincoli hanno incentivato la ricerca di nuove architetture in grado di mantenere l'immutabilità del registro distribuito, ma, allo stesso tempo, offrire throughput e latenza minori rispetto alle precedenti. Le soluzioni di terza generazione, tra cui Cardano, Polkadot e appunto Solana, si distinguono per l'adozione di protocolli ibridi, progettati per massimizzare parallelismo, risposta in tempo reale e modularità del consenso.

Capitolo 2

Panoramica dell'architettura modulare di Solana

Solana si distingue per la sua architettura modulare, dove ogni componente gioca un ruolo cruciale nel raggiungere un throughput elevato. I moduli di questa architettura sono:

2.1 Proof of History (PoH)

Un orologio globale decentralizzato che registra il tempo e l'ordine degli eventi, riducendo la necessità di comunicazioni aggiuntive per il consenso.

2.2 Tower Byzantine Fault Tolerance (Tower BFT)

Un protocollo di consenso ottimizzato per PoH, che sfrutta il Timestamping per raggiungere un accordo sui blocchi in modo rapido ed efficiente, tollerando nodi malfunzionanti.

2.3 Turbine

Un protocollo di propagazione dei blocchi, ispirato a BitTorrent, che distribuisce i dati dei blocchi in modo efficiente a migliaia di validatori utilizzando una struttura ad albero e codici di correzione degli errori.

2.4 Gulf Stream

Un protocollo di gestione delle transazioni che elimina la necessitá di un mempool tradizionale, inoltrando le transazioni direttamente ai futuri leader per un'esecuzione anticipata e una riduzione di utilizzo della memoria.

2.5 Sealevel

Un runtime parallelo per smart contract che esegue simultaneamente milioni di transazioni non conflittuali, sfruttando hardware multi-core e GPU per massimizzare la potenza computazionale.

2.6 Pipelining

Un meccanismo di ottimizzazione della validazione delle transazioni che divide il processo in fasi e le assegna a diverse unità di elaborazione, aumentando il throughput complessivo.

2.7 Cloudbreak (Accounts database)

Un sistema di memorizzazione dello stato progettata per scalare orizzontalmente con SSD, permettendo accessi concorrenti e veloci ai dati degli account, essenziali per l'esecuzione parallela.

2.8 Archivers

Un sistema di storage distribuito per la storia del ledger, che trasferisce questo compito dai validatori a nodi specializzati, garantendo la disponibilità a lungo termine dei dati storici in modo decentralizzato ed efficiente.

Capitolo 3

Analisi dettagliata dei moduli architetturali

Dopo aver introdotto l'architettura di Solana, nei prossimi capitoli parlerò in dettaglio di ogni modulo, in modo da aver chiaro il funzionamento di ciascun componente.

3.1 Proof of History (PoH)

Il Proof of History rappresenta un approccio innovativo per la sincronizzazione temporale e la verifica dell'ordine degli eventi su Solana. In questo capitolo analizzo in dettaglio le componenti tecniche alla base della sua implementazione, illustrandone il funzionamento e le modalità di integrazione con altre funzioni crittografiche della rete.

3.1.1 Funzionamento SHA-256 sequenziale

Il core del meccanismo PoH si basa sull'uso di una funzione hash, tipicamente SHA-256, eseguita in modo sequenziale. Questa operazione crea una catena di hash in cui ogni output diventa input per il calcolo successivo, garantendo così:

- **Unicità del percorso computazionale:** l'iterazione sequenziale rende impossibile saltare passaggi o parallelizzare il calcolo, creando una prova verificabile dell'ordine temporale.
- **Immutabilità:** ogni hash dipende in modo univoco da tutti quelli precedenti, rendendo ogni modifica retroattiva tecnicamente proibitiva.

- **Efficienza verificabile:** i nodi validatori possono ricalcolare la sequenza hash per verificare l'autenticità e l'ordine temporale degli eventi, senza necessità di accordi esterni.

3.1.2 Integrazione con eventi esterni (transazioni)

Il modulo PoH non opera in isolamento, bensì deve interagire con eventi esterni, quali le transazioni e altre operazioni sulla rete:

- **Timestamping delle transazioni:** ogni evento esterno viene inserito nel flusso temporale attraverso un meccanismo di timestamping, associandolo al punto esatto della sequenza hash. Questo permette una cronologia verificabile degli eventi.
- **Sincronizzazione degli input esterni:** la sequenza PoH raccoglie e integra gli input provenienti dalle transazioni garantendo che l'ordine stabilito dalla funzione hash sia mantenuto.
- **Verifica e validazione:** i nodi possono controllare che gli eventi esterni siano conformi all'ordine determinato dalla sequenza PoH, migliorando così la coerenza e l'affidabilità dell'intera rete.

L'integrazione di eventi esterni nel flusso PoH è fondamentale per mantenere una visione globale e coerente degli eventi, permettendo ad ogni transazione di essere verificata rispetto al contesto temporale predeterminato.

3.1.3 Schema di firma per binding temporale

PoH utilizza uno schema di firma digitale che lega ogni evento ad un determinato istante temporale:

- **Autenticità degli eventi:** la firma digitale, associata ad ogni hash, permette di attestare l'origine dell'evento e di verificarne l'integrità.
- **Binding crittografico:** attraverso la firma, il messaggio che costituisce un evento viene "crittograficamente" legato all'hash dell'iterazione precedente, garantendo l'univocità del legame temporale.
- **Non ripudiabilità:** l'utilizzo di chiavi crittografiche asimmetriche assicura che l'autore di un evento non possa negare in un secondo momento di averlo firmato, rafforzando l'affidabilità del sistema.

Questo schema di firma risulta determinante per la creazione di una cronologia ininterrotta e verificabile degli eventi, dove ogni blocco di dati porta con sé la prova crittografica che ne garantisce l'ordine e l'autenticità.

3.1.4 Sincronizzazione, Leadership e Tolleranza ai guasti

É importante capire come PoH contribuisce alla sincronizzazione della rete, al processo di leadership ed alla resilienza complessiva:

- **Sincronizzazione decentralizzata:** sebbene PoH generi un orologio verificabile all'interno del leader, la sua integrazione nel protocollo di consenso (Tower BFT) permette di estendere questa sincronizzazione a tutta la rete. I validatori verificano la sequenza PoH generata dal leader per accertare l'ordine degli eventi proposti nel blocco. Questa verifica, indipendente da parte di ogni validatore, assicura una visione coerente del tempo attraverso la rete, senza la necessità di una dipendenza da una singola fonte di clock esterno.

- **Ruolo del leader nella generazione PoH:** un leader viene eletto per un periodo (slot) specifico per generare la sequenza PoH e proporre un nuovo blocco di transazioni. Il leader esegue continuamente SHA-256, incorporando periodicamente gli hash delle transazioni nel flusso. La lunghezza della sequenza PoH generata, durante quello slot, fornisce una misura del tempo trascorso. Al termine, il testimone della leadership passa al validatore successivo, che continua a generare la propria sequenza PoH a partire dall'ultimo stato verificato. Questa rotazione impedisce la centralizzazione della generazione della sequenza PoH.

- **Tolleranza ai guasti e continuità della rete:** anche in presenza di ritardi di rete o partizioni temporanee, i validatori possono fare affidamento sulla sequenza PoH per verificare l'ordine degli eventi una volta che la comunicazione viene ristabilita. Poiché ogni leader genera la propria sequenza PoH, la rete può continuare a progredire anche se alcuni leader falliscono o sono temporaneamente disconnessi. Il protocollo di consenso Tower BFT, ottimizzato per PoH, sfrutta questa cronologia verificabile per raggiungere rapidamente il consenso sui blocchi, anche in condizioni di rete non ottimali. La capacità dei validatori di verificare autonomamente l'ordine degli eventi riduce la dipendenza da messaggi di coordinamento complessi, migliorando la resilienza della rete.

3.2 Tower BFT

Tower BFT rappresenta il meccanismo di consenso ibrido che integra il Proof of History (PoH) con il Proof of Stake (PoS) in un sistema che ricava ispira-

zione dai principi del Practical Byzantine Fault Tolerance (PBFT). In questo capitolo si analizzano i principi alla base di tale approccio, evidenziandone le caratteristiche chiave e i vantaggi in termini di sicurezza e scalabilità.

3.2.1 Integrazione tra PoH e PBFT

Il meccanismo Tower BFT sfrutta il flusso temporale generato da PoH per fornire una base sicura e verificabile sull'ordine degli eventi, integrando i principi del PBFT per raggiungere il consenso in presenza di possibili nodi malevoli. In particolare:

- **Utilizzo del timestamping crittografico:** il flusso PoH fornisce una sequenza inconfutabile di eventi, che viene utilizzata per evitare duplicazioni e per mantenere l'ordine temporale delle transazioni.
- **Sottoinsieme di validatori:** i validatori, scelti in maniera ponderata sulla base dello stake, eseguono il protocollo PBFT in un contesto temporale garantito dal PoH, che riduce il numero di round di comunicazione necessari per il consenso.
- **Resilienza a comportamenti malevoli:** l'architettura combinata minimizza l'impatto dei nodi compromessi, dato che il meccanismo PoH aggiunge ulteriore complessità alla possibilità di manipolazioni, integrandosi alla perfezione con i controlli PBFT.

3.2.2 Ruolo dei validatori

Il modello di consenso di Solana prevede la partecipazione di validatori il cui potere di voto è proporzionale allo stake detenuto. Questa metodologia introduce diversi vantaggi:

- **Incentivo economico:** gli stakeholder hanno un incentivo diretto a comportarsi correttamente, poiché il valore del loro investimento è legato alla sicurezza dell'intera rete.
- **Resilienza e decentralizzazione:** il sistema premia la partecipazione attiva dei nodi affidabili, riducendo il rischio di centralizzazione del potere decisionale.
- **Mitigazione degli attacchi:** poiché il peso del voto è associato allo stake, eventuali tentativi di attacco devono affrontare una barriera economica elevata, rendendo tali operazioni meno redditizie o addirittura impraticabili.

3.2.3 Meccanismo di Locking dei voti e finalità

Un aspetto peculiare di Tower BFT è il meccanismo di Locking dei voti, che contribuisce significativamente alla rapidità con cui le transazioni raggiungono la finalità. Questo meccanismo sfrutta PoH per ottimizzare il processo di voto:

- **Votazione basata su Slot e PoH:** i validatori votano su specifici slot, che rappresentano intervalli di tempo definiti dalla sequenza PoH. Invece di votare su ogni singolo blocco in modo isolato, i validatori "bloccano" i loro voti per una sequenza continua di slot, a partire dal primo blocco su cui votano.

- **Supermajority e Locking progressivo:** un blocco viene considerato finalizzato quando una supermajority di stake (superiore ai 2/3) ha votato per esso e per tutti i blocchi precedenti in una catena continua. Il meccanismo di locking fa sì che, una volta che un validatore ha votato per un certo blocco, il suo stake è implicitamente impegnato anche per i blocchi successivi, fino a quando non si verifica un evento specifico (come un fork rilevato). questo "locking progressivo" accelera il raggiungimento della finalità, poiché non è necessario un nuovo round completo di votazioni per ogni blocco successivo.

- **Finalità rapida e prevedibile:** Solana è in grado di raggiungere la finalità delle transazioni in tempi relativamente brevi (spesso entro pochi blocchi, tipicamente nell'ordine di centinaia di millisecondi).

- **Mitigazione dei problemi del PBFT:** l'integrazione con PoH aiuta a mitigare alcuni problemi legati al PBFT tradizionale. Ad esempio, la dipendenza da round di comunicazione sincroni può essere un collo di bottiglia in reti con alta latenza. PoH fornisce un ordine temporale verificabile indipendente dalla comunicazione diretta tra i nodi, riducendo la necessità di molteplici round di "pre-prepare", "prepare" e "commit" per ogni blocco. Il flusso temporale di PoH funge da base per coordinare i voti, semplificando il protocollo e migliorando la sua efficienza in ambienti distribuiti.

- **Penalità per voti incoerenti (slashing):** per incentivare un comportamento onesto e prevenire attacchi, Tower BFT implementa meccanismi di penalizzazione (slashing). I validatori che votano in modo incoerente o tentano di "doppia spesa" (votando per blocchi concorrenti nella stessa altezza) rischiano di perdere parte del loro stake.

3.2.4 Analisi degli attacchi al consenso (33% vs 51%)

La sicurezza dei protocolli di consenso si misura anche in base alla loro capacità di resistere ad attacchi di tipo Byzantine, e l'architettura Tower BFT è stata progettata per tollerare una certa percentuale di nodi malevoli:

- **Soglia del 33%:** il sistema rimane sicuro fino a quando meno di un terzo dei nodi sono compromessi. Tale soglia è cruciale per garantire che un numero sufficiente di validatori onesti possa impedire la corruzione del consenso.
- **Soglia del 51%:** un attacco che raggiunga o superi il 51% dello stake totale mette a rischio l'integrità dell'intero sistema, permettendo potenzialmente il controllo della rete.

Un confronto tra queste due soglie evidenzia quanto segue:

1. **Resilienza al 33%:** finché la percentuale di validatori malevoli resta inferiore al 33%, il meccanismo Tower BFT garantisce un'elevata robustezza, grazie ai controlli incrociati e all'ordine temporale definito dal PoH.
2. **Criticità oltre il 51%:** il superamento del 51% dello stake potrebbe determinare un broken-chain attack, in cui il potere di voto concentrato nei validatori malevoli potrebbe alterare l'ordine degli eventi, riscrivere la cronologia o compromettere la finalità delle transazioni. Questo scenario espone la necessità di meccanismi di penalizzazione e di incentivi correttivi per prevenire la centralizzazione di potere in singoli nodi.

3.3 Turbine

Turbine è il protocollo di propagazione dei dati di Solana, progettato per distribuire rapidamente i blocchi su una rete di migliaia di nodi con latenza minima e uso efficiente della banda. In questo capitolo analizzo in dettaglio le componenti tecniche di Turbine, illustrandone il funzionamento, le ottimizzazioni di affidabilità e il contributo alla scalabilità della rete.

3.3.1 Suddivisione in pacchetti e trasmissione a fanout

Il primo passo di Turbine è frammentare ogni blocco in pacchetti di dimensione fissa (tipicamente 64 KiB).

- **Chunking dei blocchi:** il blocco viene suddiviso in più chunk indipendenti, ognuno numerato e identificato in modo univoco.
- **Albero di fanout:** il leader invia ciascun chunk a un sottoinsieme di validatori (primo livello), i quali a loro volta lo ritrasmettono al livello successivo, e così via, in una struttura ad albero multi-livello.
- **Complessità logaritmica:** con un fanout di f nodi per livello, la propagazione raggiunge N nodi in circa $\log_f N$ salti, riducendo drasticamente il numero di connessioni dirette necessarie.

3.3.2 Codici di correzione d'errore e robustezza

Per garantire che i validatori ricostruiscano l'intero blocco anche in presenza di perdita di pacchetti, Turbine integra codici di correzione d'errore:

- **Reed-Solomon encoding:** prima della trasmissione, i chunk sono codificati con schemi Reed-Solomon, permettendo la ricostruzione del blocco anche se fino al 33% dei pacchetti è andato perduto.
- **Ridondanza controllata:** il leader può inviare pacchetti supplementari a nodi selezionati per migliorare l'affidabilità nei percorsi più soggetti a perdita.
- **Verifica integrità:** ogni chunk include checksum crittografici per consentire ai validatori di rilevare e richiedere eventuali pacchetti corrotti o mancanti.

3.3.3 Ottimizzazione della banda e latenza

Turbine ottimizza l'uso della rete e minimizza la latenza di propagazione dei blocchi grazie a:

- **UDP streaming:** utilizzo di UDP anziché TCP per evitare handshake e back-off di congestione.
- **Adaptive fanout:** il numero di peer per livello può essere adattato dinamicamente in base alla topologia di rete e alla latenza misurata.
- **Parallelismo multi-threaded:** i validatori processano e ritrasmettono i pacchetti in thread separati, evitando colli di bottiglia in I/O.

3.3.4 Contributo alla scalabilità e resilienza di rete

L'integrazione di Turbine nel sistema di Solana garantisce:

- **Scalabilità su larga scala:** la complessità $\mathcal{O}(\log N)$ permette di crescere a decine di migliaia di validatori senza aumenti esponenziali di latenza.
- **Tolleranza ai guasti di rete:** grazie ai codici di correzione e al fanout multilivello, la rete rimane operativa anche con perdite significative di pacchetti.
- **Throughput elevato:** riducendo l'overhead di trasmissione e sfruttando UDP, Turbine contribuisce a sostenere il throughput di decine di migliaia di transazioni al secondo.

3.4 Gulf Stream

Il modulo Gulf Stream di Solana sostituisce la classica mempool con un sistema dinamico di forwarding delle transazioni verso i validatori designati per i futuri slot. Questo approccio permette di ridurre drasticamente la latenza di ordinamento e di alleggerire il carico di memoria sui nodi. In questo capitolo, analizzo in dettaglio il funzionamento tecnico di Gulf Stream, il suo effetto sulla pipeline di esecuzione e la resilienza che conferisce alla rete.

3.4.1 Funzionamento di forwarding anticipato

- **Predizione del leader:** grazie alla sequenza PoH, ogni nodo conosce in anticipo l'identità del validatore che avrà la leadership nei prossimi slot.
- **Invio diretto delle transazioni:** i client inoltrano le transazioni direttamente al validatore futuro, firmandole con un recente hash PoH come nonce temporale.
- **Validazione preliminare:** il validatore ricevente verifica immediatamente la validità delle firme e dei requisiti di account, preparando le transazioni prima ancora di proporle in un blocco.

3.4.2 Eliminazione della mempool e cache locale

- **Nessuna mempool pubblica:** non esiste più un'area di attesa globale; ogni validatore mantiene solo una cache delle transazioni attese per il proprio slot.
- **Caching distribuito:** le transazioni circolano attraverso un grafo di caching che permette ai validatori vicini di condividere rapidamente liste di transazioni non ancora incluse.
- **Pulizia automatica:** al termine di uno slot, le transazioni non elaborate vengono scartate o re-inoltrate automaticamente ai leader successivi.

3.4.3 Vantaggi prestazionali

- **Riduzione della latenza:** le transazioni possono essere eseguite non appena inizia lo slot, anziché attendere la conferma dell'inclusione in mempool.
- **Maggiore throughput:** il caricamento anticipato consente di sfruttare meglio le risorse di calcolo e di rete, incrementando il numero di transazioni processabili per secondo.
- **Minore contesa di risorse:** poiché ogni validatore riceve solo il proprio sottoinsieme di transazioni, si riducono i conflitti di accesso e gli overhead di sincronizzazione.

3.4.4 Tolleranza ai guasti e continuità della rete

- **Resilienza al fallimento del leader:** se un validatore leader non è raggiungibile, le transazioni vengono automaticamente inoltrate al successivo leader previsto.
- **Mitigazione del congestionamento:** in caso di picchi transazionali, Gulf Stream bilancia il carico distribuendo le transazioni su più validatori futuri.
- **Recupero da partizioni di rete:** le cache locali mantengono le transazioni in attesa fino al ripristino della connettività, assicurando che nessun client perda le proprie operazioni.

3.5 Sealevel

Sealevel è il motore di esecuzione parallela degli smart contract di Solana, progettato per sfruttare appieno l'hardware moderno e permettere un throughput elevato. In questo capitolo, approfondisco la sua architettura, i modelli di concorrenza adottati e le implicazioni per la scalabilità dell'intera rete.

3.5.1 Motivazioni ed architettura

L'esecuzione degli smart contract tradizionalmente segue un modello sequenziale, che limita drasticamente il throughput. Sealevel rompe questo paradigma introducendo un modello completamente parallelo:

- **Parallellismo basato sugli account:** gli smart contract su Solana dichiarano in anticipo tutti gli account che andranno a leggere o modificare. Questo consente al runtime Sealevel di schedulare in parallelo tutte le istruzioni che non si sovrappongono negli account.
- **Approccio "Compute Budgeting":** ogni transazione specifica un "budget" computazionale, evitando operazioni che potrebbero causare congestione e bloccaggio dell'interprete.
- **Indipendenza dallo stato globale:** le istruzioni vengono isolate in base agli account coinvolti, riducendo i conflitti e migliorando l'efficienza.

Questa architettura permette l'esecuzione simultanea di migliaia di smart contract in una singola unità di tempo, abilitando una scalabilità orizzontale reale.

3.5.2 Modello di concorrenza

Sealevel implementa un sistema di concorrenza deterministica che si basa su due concetti fondamentali: il controllo degli accessi agli account e la gestione predittiva delle dipendenze.

Analisi statica delle dipendenze

Quando una transazione viene proposta, viene prima analizzata per determinare:

- Gli account che verranno letti (read set).
- Gli account che verranno scritti (write set).

Sealevel costruisce un grafo delle dipendenze tra transazioni e identifica i conflitti di accesso, consentendo l'esecuzione parallela solo di quelle che operano su insiemi disgiunti.

Parallel Scheduler ottimizzato

Il runtime Sealevel implementa uno scheduler parallelo che sfrutta al massimo i core CPU disponibili:

- Le istruzioni vengono eseguite in thread multipli, finché non emergono conflitti nel write set.
- L'accesso concorrente in lettura è permesso, ma ogni conflitto di scrittura viene gestito serializzando l'esecuzione delle istruzioni coinvolte.

3.5.3 Compatibilità con il modello Solana

L'integrazione di Sealevel nel sistema Solana è resa possibile dal particolare modello di "Account-based architecture" e dalla programmazione deterministica degli smart contract:

- **Programmazione via BPF (Berkeley Packet Filter):** gli smart contract scritti per Sealevel sono compilati in BPF, un linguaggio sicuro ed ad alte prestazioni che può essere eseguito in ambienti isolati.
- **Chiamate incrociate asincrone:** Sealevel consente a un programma di invocare altri programmi in maniera asincrona, pur mantenendo la sicurezza dell'esecuzione.
- **Isolamento della memoria:** l'esecuzione dei contratti è Sandboxed: ogni programma ha accesso solo agli account e alla memoria dichiarati inizialmente.

3.5.4 Vantaggi

- **Scalabilità orizzontale:** l'esecuzione parallela permette di scalare il throughput linearmente con il numero di core disponibili.
- **Riduzione della latency:** le transazioni indipendenti vengono eseguite simultaneamente, riducendo drasticamente il tempo medio di conferma.
- **Efficienza computazionale:** il modello di "Compute Budgeting" evita l'utilizzo eccessivo delle risorse, permettendo un uso efficiente dell'hardware.

3.5.5 Limiti

- **Conflitti di scrittura:** Le transazioni che modificano lo stesso account restano soggette a serializzazione.
- **Programmazione più complessa:** Gli sviluppatori devono dichiarare in anticipo tutti gli account usati, il che rende la logica dei programmi più rigida.
- **Debugging e Testing:** La concorrenza introduce sfide aggiuntive per il testing predittivo di smart contract complessi.

3.6 Pipelining

Il pipelining in Solana è un meccanismo che sovrappone e parallelizza le diverse fasi del processamento delle transazioni, riducendo drasticamente la latenza complessiva e massimizzando l'utilizzo delle risorse hardware. In questo capitolo, approfondisco il funzionamento del pipelining, i suoi componenti principali, i benefici in termini di throughput e i possibili limiti.

3.6.1 Funzionamento delle pipeline interne

Solana suddivide l'elaborazione di blocchi e transazioni in una serie di stage distinti, ciascuno dei quali può operare in parallelo su batch differenti di dati:

1. **Ricezione dei pacchetti (fetch)**: i nodi validatori ricevono i pacchetti UDP propagati da Turbine e ne eseguono un primo filtraggio (checksum, autenticità).
2. **Verifica delle firme (signature verification)**: tramite batch su GPU o CPU, le firme Ed25519 vengono verificate in parallelo per centinaia di transazioni contemporaneamente.
3. **Scheduling e analisi delle dipendenze (banking stage)**: il runtime esamina le transazioni per identificare conflitti di accesso allo stato (account read/write), raggruppando quelle non conflittuali.
4. **Esecuzione degli smart contract (execution)**: sfruttando Sealevel, ogni gruppo di transazioni indipendenti viene eseguito simultaneamente su core CPU e, quando possibile, su istruzioni SIMD delle GPU.
5. **Aggiornamento dello stato (write back)**: le modifiche agli account vengono scritte sul database Cloudbreak in modalità Memory-mapped, con operazioni I/O parallele su SSD.

6. **Broadcast dei risultati (broadcast)**: i risultati finali (nuovo stato e firme di conferma) vengono propagati ai peer per completare il consenso Tower BFT.

Ogni stage lavora su un batch indipendente: mentre uno stage elabora il batch n, il successivo può già iniziare a processare il batch $n-1$, creando una catena di produzione continua che sfrutta al massimo la pipeline.

3.6.2 Vantaggi

- **Riduzione della latenza:** il pipelining elimina i tempi morti tra le fasi, permettendo un tempo di blocco inferiore al secondo anche in condizioni di rete non ideali.
- **Parallelismo a più livelli:** ogni fase gira in parallelo su thread e core separati, garantendo un elevato throughput sostenuto.
- **Basso overhead di contesa:** la suddivisione in batch e la pre-analisi delle dipendenze riducono drasticamente i conflitti di accesso allo stato, migliorando la scalabilità orizzontale.
- **Scalabilità elastica:** aggiungendo risorse (core CPU, GPU, SSD), la pipeline scala automaticamente, supportando l'aumento del carico transazionale senza riprogettare l'architettura.

3.6.3 Limiti

- **Bilanciamento dei stage:** uno stage troppo lento (ad esempio operazioni I/O su SSD saturi) può ridurre l'efficienza complessiva.
- **Gestione degli errori:** rollback e retry di singoli batch in caso di fallimenti devono essere gestiti con attenzione per evitare inconsistenze di stato.
- **Complessità implementativa:** l'overhead di organizzazione delle pipeline e di sincronizzazione delle code introduce complessità nel codice del validator.

3.6.4 Contributo alla Scalabilità di Rete

Il Pipelining rappresenta uno degli elementi principali per il modello "L1-native scaling" adottato da Solana:

- Consente di mantenere elevato "per-core utilization", massimizzando le prestazioni offerte dall'hardware disponibile.
- Riduce drasticamente i tempi di "end-to-end processing" di ogni transazione, permettendo a Solana di avvicinarsi ai 5.000–10.000 TPS effettivi in mainnet-beta e di scalare fino a decine di migliaia di TPS su infrastrutture ottimizzate.
- Si integra perfettamente con PoH, Gulf Stream e Cloudbreak, fornendo un flusso continuo di dati e transazioni che scorre attraverso la rete senza strozzature significative.

Capitolo 4

Cloudbreak (Accounts database)

Cloudbreak é un database progettato per gestire milioni di account in maniera veloce e scalabile. Il compito principale di Cloudbreak é tenere traccia dello stato degli account. Ogni volta che viene eseguita una transazione, i profili coinvolti vengono letti e aggiornati nel minor tempo possibile. In questo capitolo, entreró in dettaglio su come Cloudbreak riesce a raggiungere questi obiettivi tramite una serie di tecniche:

- **Memory-mapped files**: anziché caricare tutto in memoria o scrivere ogni volta su disco, Cloudbreak lascia che sia il sistema operativo a gestire la memoria, mappando direttamente i file su disco in modo da ridurre i tempi di accesso.
- **Copy-on-write**: quando un account viene aggiornato, i dati non vengono sovrascritti ma copiati in una nuova posizione, cosí da evitare conflitti e mantenere sempre una versione coerente dello stato.
- **Indicizzazione efficiente**: ogni account ha chiave pubblica che viene indicizzata per trovare rapidamente la posizione dei suoi dati nel database.

4.1 Funzionamento di Cloudbreak

Per comprendere meglio Cloudbreak, é utile analizzare il suo funzionamento durante l'esecuzione di un blocco. Ogni validatore lo utilizza per leggere lo stato degli account coinvolti in una transazione, aggiornare tali dati e garantire che il risultato sia consistente e persistente nel tempo, applicando i seguenti passaggi:

1. **Caricamento dello stato iniziale**: all'inizio dell'elaborazione di un blocco, il validatore accede ad uno snapshot dello stato corrente tramite memory-mapped files, in modo da avere una base di riferimento immutabile per il passo successivo.

2. **Esecuzione parallela delle transazioni**: grazie al runtime Sealevel, più transazioni vengono eseguite in parallelo. Cloudbreak fornisce un accesso concorrente lock-free alla memoria, permettendo a ciascun thread di leggere i dati necessari senza interferenze.

3. **Aggiornamento dello stato**: se una transazione modifica lo stato di un account, viene creato un nuovo record per l'account aggiornato, cosí che le altre transazioni in esecuzione continuino a leggere la versione originale.

4. **Batching delle scritture**: al termine del blocco, le modifiche vengono raccolte in un batch e scritte in modo atomico sui dischi SSD.

4.2 Pulizia e gestione dello spazio

Cloudbreak include una serie di meccanismi per mantenere il database leggero ed efficiente:

- **Flushing**: trasferisce i dati temporanei dalla RAM al disco.
- **Cleaning**: elimina vecchie versioni dei dati non più utilizzate.
- **Shrinking**: riduce le dimensioni dei file rimuovendo porzioni inutilizzate.
- **Purging**: cancella completamente i dati legati a rami della blockchain che sono stati abbandonati.

4.3 Scalabilità orizzontale, Consistenza e Resilienza di Cloudbreak

Oltre al design di base per l'accesso concorrente allo stato, é importante comprendere come Cloudbreak assicuri la scalabilità orizzontale, mantenga la consistenza dei dati e tolleri guasti hardware o di rete:

- **Striping RAID 0 e mappatura in memoria:** Cloudbreak organizza i dati degli account su più SSD in striping RAID 0, mappando ciascun file su memoria virtuale tramite mmap. Questo approccio permette di distribuire le richieste di I/O tra tutti i dischi disponibili, riducendo drasticamente la latenza di lettura e scrittura ed aumentando linearmente il throughput con l'aggiunta di ogni nuovo SSD.

- **Accesso concorrente e lock-free reads:** grazie alla mappatura in memoria, le operazioni di lettura possono essere eseguite in modalità lock-free: ogni thread legge direttamente dal proprio segmento di memoria mappata senza competere per mutex o lock a livello di file. Le operazioni di scrittura vengono coordinate tramite buffer di batching che raccolgono più aggiornamenti e li scrivono periodicamente sui dischi, minimizzando la contesa e l'overhead di sincronizzazione tra thread.

- **Consistency model:** all'inizio di ogni blocco, il validatore fissa un "bookmark" sullo stato corrente (snapshot), garantendo che tutte le transazioni incluse in quel blocco leggano uno stato immutabile. Le modifiche vengono applicate in modo atomico solo al termine dell'elaborazione del blocco, evitando letture incoerenti o "dirty reads" durante l'esecuzione parallela su Sealevel.

- **Tolleranza ai guasti dei dischi:** pur non includendo ridondanza hardware completa, Cloudbreak sfrutta il fatto che ogni account é replicato su più validatori: in caso di guasto di un SSD, i dati possono essere recuperati leggendo il medesimo snapshot dallo storage di un altro validatore. Inoltre, i validatori possono "ricostruire" lo stato mancato eseguendo nuovamente le transazioni dal ledger a partire dall'ultimo snapshot sano.

- **Recupero incrementale e Compaction:** per evitare che i file mappati crescano indefinitamente, Cloudbreak effettua periodicamente operazioni di "Compaction", dove raccoglie i delta di stato in un nuovo file snapshot, rilasciando lo spazio su disco occupato dalle vecchie versioni. Questo processo é eseguito in background ed assicura che lo storage rimanga efficiente anche dopo migliaia di blocchi.

4.4 Archivers

Gli Archivers sono nodi specializzati incaricati di conservare la storia completa del ledger, alleggerendo i validatori dal carico associato alla memoriz-

zazione persistente dei dati. In questo capitolo, entrerò in dettaglio sulle componenti crittografiche e di sistema che governano il loro funzionamento, illustrando i meccanismi di frammentazione, verifica e resilienza.

4.4.1 Funzionamento: Erasure Coding e Proofs of Replication

- **Frammentazione del ledger:** ogni blocco storico viene suddiviso in frammenti di dimensioni fisse e codificato con "Erasure Codes" (tipicamente Reed–Solomon). Questo permette di ricostruire l'intero blocco anche se solo una parte dei frammenti è disponibile.
- **Distribuzione dei frammenti:** i frammenti codificati vengono assegnati in modo pseudo-casuale agli Archivers mediante un meccanismo deterministico derivato da PoH, garantendo un'equa distribuzione e riducendo i rischi di centralizzazione.
- **Proofs of Replication (PoRep):** ogni Archiver, periodicamente, genera una prova crittografica che attesta di possedere i frammenti a lui assegnati. Le PoRep si basano su "challenge" derivate dalla sequenza PoH e su tecniche di "merkleization" dei dati (trasformare un insieme di dati in un Merkle tree).

4.4.2 Integrazione con PoH

- **Temporalità delle challenge:** le challenge PoRep sono derivate dall'hash di posizioni specifiche nella catena PoH, garantendo che un Archiver non possa pre-computare o riutilizzare prove.
- **Verifica distribuita:** i validatori raccolgono le PoRep inviate dagli Archivers e ne validano l'autenticità ricontrollando la catena PoH e la merkle root dei frammenti.
- **Sincronizzazione e incentivi:** solo gli Archivers che presentano PoRep corrette entro una finestra temporale stabilita (basato su PoH) ricevono ricompense in SOL, incentivando la partecipazione e la corretta conservazione dei dati.

4.4.3 Verifica periodica e Gestione delle contromisure

- **Controlli randomizzati:** le challenge non seguono un calendario prevedibile, ma sono pseudo-randomizzate attraverso PoH, aumentando la difficoltà di attacco mirato.
- **Penalità e ricompense:** Archivers inattivi o che falliscono le PoRep subiscono slashing parziale dello stake o esclusione temporanea dalla rete, mentre quelli affidabili guadagnano premi proporzionali alle proof inviate.
- **Rotazione dei frammenti:** il sistema riassegna, periodicamente, i frammenti agli Archivers per mitigare l'accumulo di dati non utilizzati e per ridurre il rischio di compromissione mirata.

4.4.4 Tolleranza ai guasti e Disponibilità

- **Ridondanza:** Grazie agli "Erasure Codes", la rete può tollerare la perdita fino a k frammenti per blocco (dove k è scelto secondo il livello di ridondanza configurato) senza compromettere la capacità di recupero.
- **Recupero dinamico:** In caso di crash o malfunzionamento di un Archiver, gli altri nodi specializzati restituiscono rapidamente i frammenti mancanti, supportati dal meccanismo PoRep e dai challenge PoH.
- **Scalabilità orizzontale:** È possibile aggiungere nuovi Archivers "on the fly": una volta sincronizzati con l'ultimo stato PoH, ricevono i frammenti e iniziano subito a generare PoRep, estendendo linearmente la capacità di archiviazione dello storico.

4.4.5 Contributo alla scalabilità della rete

Gli Archivers permettono di mantenere una blockchain di dimensioni crescenti senza sovraccaricare i validatori, garantendo al contempo elevata disponibilità e integrità dei dati storici. La sinergia con PoH e PoRep assicura una conservazione decentralizzata, verificabile e economicamente incentivata.

Capitolo 5

Algoritmi crittografici principali

In questo capitolo, vengono analizzati gli algoritmi crittografici principali che supportano il funzionamento e la sicurezza di Solana. In particolare, viene esaminata la pipeline di verifica transazionale, che sfrutta Ed25519 per le firme digitali e SHA-256 per il Proof of History, e si approfondisce il meccanismo di Proof of Replication che, integrando tecniche basate su SeDRAM, si propone come strumento di difesa contro gli attacchi Sybil.

5.1 Pipeline di verifica transazionale: Ed25519 per firme digitali e SHA-256 per PoH

La sicurezza e velocità della rete Solana si fondano su una pipeline crittografica accuratamente progettata, in cui due algoritmi giocano un ruolo fondamentale:

5.1.1 Ed25519 per firme digitali

- **Efficienza e sicurezza:** Ed25519 è una variante dell'algoritmo a curva ellittica progettata per garantire alte prestazioni ed un elevato grado di sicurezza. Grazie alla sua struttura, permette di generare firme digitali che sono sia compatte che verificabili in maniera rapida.
- **Non ripudiabilità e integrità:** l'impiego di Ed25519 assicura che ogni messaggio e transazione siano firmati in modo univoco, impedendo a un partecipante di negare in seguito la propria partecipazione e garantendo l'integrità dei dati trasmessi.

5.1.2 SHA-256 per il PoH

- **Costruzione di una sequenza temporale:** SHA-256 viene eseguito in modalità iterativa per generare una catena di hash che costituisce la base temporale di PoH. Ogni output della funzione hash diventa input per il successivo ciclo, creando una sequenza ininterrotta e verificabile.

- **Resistenza alle collisioni:** la robustezza di SHA-256, nota per la sua bassa probabilità di collisione, garantisce che ogni step della sequenza sia univoco e resistano ad eventuali tentativi di manipolazione. Questo rende l'ordine degli eventi irreversibile e verificabile in maniera autonoma dai validatori.

5.2 Proof of Replication (PoRep): SeDRAM e Resistenza agli attacchi Sybil

Il meccanismo di Proof of Replication (PoRep) rappresenta un ulteriore livello di sicurezza e verifica, volto a dimostrare che i dati sono stati replicati in maniera adeguata nei nodi della rete. In questo contesto, il supporto tecnologico fornito da SeDRAM, unitamente a strategie di mitigazione contro gli attacchi Sybil, gioca un ruolo cruciale.

5.2.1 SeDRAM ed Ottimizzazione della replicazione

- **Memoria ad alte prestazioni:** SeDRAM (Synchronous Dynamic Random Access Memory) viene impiegata per ottimizzare il processo di replicazione dei dati, garantendo tempi di accesso ridotti e una gestione efficiente della memoria. Questo è fondamentale per validare rapidamente la presenza e l'integrità delle repliche.

- **Verifica dell'integrità dei dati:** grazie a SeDRAM, il sistema è in grado di effettuare operazioni di verifica ripetute ed ad alta velocità, assicurando che ogni replica mantenga la stessa struttura ed integrità del dato originale.

5.2.2 Resistenza agli attacchi Sybil

- **Definizione degli attacchi Sybil:** gli attacchi Sybil si verificano quando un singolo utente crea numerose identità false per ottenere un controllo sproporzionato sulla rete. Questo tipo di attacco può compromettere la sicurezza del sistema, se non adeguatamente contrastato.

- **Mitigazione tramite PoRep:** il processo di Proof of Replication, unito alla gestione efficiente della memoria tramite SeDRAM, impone un elevato costo computazionale e di risorse per la creazione di repliche false. Quidni, replicare e dimostrare la presenza dei dati in maniera fraudolenta richiede investimenti significativi, rendendo l'attacco meno vantaggioso dal punto di vista economico.
- **Integrazione con altri meccanismi di sicurezza:** PoRep si integra con il protocollo di consenso e altri strati di sicurezza per rafforzare la resilienza della rete contro la formazione di nodi Sybil.

Capitolo 6

Gestione chiavi e Sicurezza della rete

La sicurezza di Solana dipende sia dagli algoritmi crittografici sottostanti, sia dalla gestione efficace delle chiavi e dalla difesa contro attacchi di rete. In questo capitolo, esplorerò le soluzioni adottate in termini di architettura delle chiavi, firme crittografiche collaborative (Threshold Signatures), e tecniche avanzate per la rotazione sicura delle chiavi e la protezione contro attacchi di tipo DDoS.

6.1 Chiavi gerarchiche

Solana adotta un'architettura di chiavi pubbliche "gerarchiche" per facilitare la gestione della sicurezza, soprattutto in applicazioni ad alto volume. Il modello è ispirato a BIP32 (Bitcoin Improvement Proposal), ma adattato all'ecosistema Solana. Le caratteristiche principali includono:

- **Derivazione deterministica:** le chiavi derivate da una chiave master possono essere rigenerate in qualsiasi momento, purché si conosca la chiave iniziale e il percorso di derivazione, migliorando la portabilità e la sicurezza.
- **Isolamento tra livelli:** ogni livello della gerarchia può essere isolato, limitando l'impatto di una potenziale compromissione su sottoinsiemi specifici di chiavi.
- **Gestione di accessi granulari:** tramite percorsi di derivazione specifici, è possibile creare sotto-chiavi con permessi limitati, ideali per deleghe o sessioni temporanee.

6.2 Threshold Signatures

Per migliorare la robustezza contro attacchi e guasti, Solana può impiegare "Threshold Signatures" basate su schemi crittografici come Shamir's Secret Sharing o protocolli compatibili con Ed25519, così da permettere ad un insieme di validatori o nodi fidati di firmare transazioni in modo distribuito.

- **Collaborazione sicura:** una firma é valida solo quando almeno t su n partecipanti collaborano, aumentando la resilienza a compromissioni parziali.
- **Assenza di single point of failure:** nessun partecipante possiede l'intera chiave privata, riducendo il rischio di compromissione totale.
- **Riduzione del carico computazionale:** alcune implementazioni permettono aggregazione e compressione delle firme, migliorando l'efficienza delle transazioni su larga scala.

6.3 Rotazione delle chiavi

Solana supporta rotazioni sia automatiche (via smart contract) sia manuali per i wallet, account e validatori. Le motivazioni principali includono:

- **Prevenzione da compromissioni prolungate:** una chiave potenzialmente compromessa, ma non ancora sfruttata può essere neutralizzata grazie alla rotazione.
- **Auditabilità e accountability:** i meccanismi di aggiornamento delle chiavi sono tracciati on-chain, rendendo possibile le veriche formali.
- **Integrazione con meccanismi multi-sig e wallet custodial:** questo facilita l'automazione di processi di sicurezza anche in ambienti aziendali.

6.4 Mitigazione DDoS con Rate-Limiting crittografico

Essendo che gli attacchi DDoS (Distributed Denial of Service) rappresentano una minaccia significativa per la blockchain, Solana adotta un sistema di "rate-limiting crittografico", utile per mantenere un elevato throughput anche in presenza di tentativi malevoli di congestione, che opera sui seguenti principi:

- **Proof-of-Stake come filtro economico:** ogni nodo che invia transazioni deve dimostrare il possesso di stake. Questo serve come barriera economica per ridurre spam e flood.
- **Verifica crittografica del volume:** il sistema può associare, ad ogni account, limiti transazionali determinati dal volume di firma e dallo stake sottostante. Se un attore supera tale soglia, viene temporaneamente penalizzato o isolato.
- **Hardware-enforced quotas:** alcuni validatori possono adottare rate limiter a livello di rete per bloccare in anticipo pacchetti malformati o duplicati.

Capitolo 7

Crittografia applicata: smart contract e NFT

In questo capitolo, analizzo le applicazioni della crittografia all'interno del sistema Solana, con particolare riferimento a smart contract e NFT: l'uso delle Verifiable Delay Functions (VDF) come elemento computazionale nei contratti intelligenti, lo standard Solana Program Library (SPL) Token con supporto per metadati cifrati, e l'integrazione progressiva dei ZK-SNARKs come strumento per abilitare la privacy on-chain.

7.1 Sfida VDF nei Contratti Intelligenti su Solana

Le VDF sono funzioni computazionalmente intensive, progettate per richiedere un determinato tempo sequenziale per essere calcolate, ma facilmente verificabili. Su Solana, l'integrazione delle VDF è considerata un possibile strumento per:

- **Incentivare on-chain randomness verificabile:** le VDF sono utilizzate come meccanismo per generare numeri casuali resistenti alla manipolazione.
- **Limitare front-running:** l'introduzione di un ritardo computazionale obbligatorio mitiga i rischi derivanti dall'osservazione delle transazioni mempool e successive manipolazioni da parte di validatori opportunistici.

- **Proof-of-Time on-chain:** le VDF possono rappresentare un'integrazione interessante con PoH per task contrattuali che richiedano latenza verificabile indipendente.

Le VDF su Solana possono essere implementate via smart contract oppure attraverso moduli off-chain collegati via syscall e verificati attraverso SHA-256 o gruppi RSA modulari.

7.2 Standard SPL Token e Metadati cifrati

Lo standard SPL definisce le specifiche per token fungibili e non fungibili. Oltre alla semplice rappresentazione di bilanci e identificatori, lo standard supporta estensioni crittografiche, tra cui i metadati cifrati.

7.2.1 Metadati cifrati

- **Crittografia simmetrica AES-GCM:** gli sviluppatori possono cifrare metadati NFT utilizzando AES-GCM, assicurando sia confidenzialità che integrità (tramite tag di autenticazione) prima di caricare i dati on-chain o su IPFS.
- **Gestione chiavi con derived addresses:** l'accesso ai contenuti cifrati può essere gestito tramite "Program-derived addresses" (PDAs), le quali possono fungere da proxy crittografici autorizzati all'accesso delle chiavi simmetriche.

Questa struttura consente un livello superiore di privacy e controllo, utile in applicazioni come musica, arte, giochi, certificati o documentazione legale su blockchain.

7.3 ZK-SNARKs per transazioni private

Sebbene Solana non abbia inserito nativamente un supporto full-stack per ZK-SNARKs (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge), si stanno sviluppando librerie e protocolli compatibili per l'esecuzione di "circuiti zk" all'interno di smart contract personalizzati. Le caratteristiche principali sono:

- **Privacy su conti e trasferimenti:** l'introduzione di ZK-SNARKs può permettere il trasferimento di token o NFT senza rivelare mittente, destinatario o quantità, garantendo al contempo la validità della transazione.

- **ZK-Rollup compatibili:** layer costruiti sopra Solana possono delegare la verifica di centinaia di transazioni off-chain ad un'unica prova SNARK verificata on-chain.
- **Ottimizzazione computazionale:** grazie alla struttura ad alta parallelizzazione di Solana, è possibile prevedere l'accelerazione della verifica SNARK tramite GPU, permettendo l'uso in tempo reale di microtransazioni private.

Bibliografia

[1] A. Yakovenko, *Solana: A new architecture for a high performance blockchain*, 2018. Disponibile su: `https://solana.com/solana-whitepaper.pdf`

[2] Solana Documentation, 2023. Disponibile su: `solana.com/docs`

[3] Solana News. Disponibile su: `solana.com/it/news`

[4] M. Castro e B. Liskov, *Practical Byzantine Fault Tolerance*, in OSDI, 1999. Disponibile su: `https://pmg.csail.mit.edu/papers/osdi99.pdf`

[5] D. J. Bernstein e T. Lange, *High-speed high-security signatures*, 2012. Disponibile su: `https://ed25519.cr.yp.to/`

[6] National Institute of Standards and Technology (NIST), *FIPS PUB 180-4: Secure Hash Standard (SHS)*, 2015. Disponibile su: `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf`

[7] Protocol Labs, *Proof-of-Replication: Scalable Decentralized Storage*, 2017. Disponibile su: `https://filecoin.io/proof-of-replication.pdf`

[8] Project Serum, *Anchor Book - Solana Smart Contract Framework*, 2022. Disponibile su: `https://book.anchor-lang.com/`

[9] Jump Crypto, *Firedancer: A High Performance Validator for Solana*, 2023. Disponibile su: `https://jumpcrypto.com/firedancer/`

[10] Protocol Labs, *The Future of Zero Knowledge Proofs*, 2023. Disponibile su: `https://www.protocol.ai/protocol-labs-the-future-of-zk-proofs.pdf`

[11] D. Boneh et al., *ZK-SNARKs: Scalable, Transparent, and Post-Quantum Secure Proofs*, 2018. Disponibile su: `https://eprint.iacr.org/2018/046.pdf`