Appunti Sicurezza dei dati

Corso di Sicurezza dei Dati

7 ottobre 2025

Indice

1 Introduzione e Caratterizzazione dei Sistemi Distribuiti

Definizione 1 (Sistema Distribuito)**.** *Un Sistema Distribuito è un insieme di componenti software, localizzati su computer interconnessi in rete, che comunicano e coordinano le loro azioni esclusivamente tramite lo scambio di messaggi.*

1.1 Aspetti Fondamentali

La principale sfida nella progettazione di questi sistemi è l'assenza di un controllo centrale e di un orologio globale.

- **Concorrenza**: I processi operano contemporaneamente.
- **Assenza di Clock Globale**: La nozione di "tempo assoluto" non esiste. La sincronizzazione è solo tramite messaggi.
- **Fallimenti Indipendenti**: Ogni componente può fallire indipendentemente dagli altri, rendendo la tolleranza ai guasti cruciale.

1.2 Modello di Interazione (Tempificazione)

Il modello di interazione definisce le ipotesi sui tempi di esecuzione e comunicazione.

- **Sistema Sincrono**: Si assumono e si conoscono limiti precisi (Upper e Lower Bound) per:
  1. Tempo di esecuzione di ogni passo di elaborazione.
  2. Tempo di consegna dei messaggi.
  3. Deriva degli orologi locali (Clock Drift Rate).
- **Sistema Asincrono**: Non esistono limiti noti o garantiti. Molti sistemi reali (es. Internet, reti basate su TCP/IP) si modellano come asincroni.

1.3 Modello dei Fallimenti

Si definiscono tre stati causali di malfunzionamento:

$$\underbrace{\text{Guasto (Fault)}}_{\text{Causa Originaria}} \longrightarrow \underbrace{\text{Errore (Error)}}_{\text{Stato Intermedio}} \longrightarrow \underbrace{\text{Fallimento (Failure)}}_{\text{Scostamento dal Comportamento Atteso}}$$

2 Il Problema del Consenso Distribuito (C)

Il problema del Consenso (C) richiede che N processi (o nodi) convergano e decidano su un unico valore d.

Tabella 1: Tassonomia dei Tipi di Fallimento

Tipo di Fallimento	Descrizione
Crash	Il processo interrompe le sue azioni e si arresta.
Fail Stop	Il processo si arresta e il suo fallimento è immediatamente rilevabile dagli altri.
Omissione (Omision)	Mancata esecuzione di un'azione attesa (es. perdita di un messaggio).
Temporale (Timing)	Mancato rispetto di una scadenza (solo in sistemi sincroni).
Arbitrario / Bizantino	Il processo si comporta in modo completamente imprevedibile, arbitrario o malizioso, potendo inviare messaggi contraddittori.

2.1 Requisiti di Consenso

Un algoritmo risolve il problema del consenso se i processi corretti soddisfano le seguenti proprietà:

1. **Validità (Validity)**:

$$(\forall p_i \in \text{corretti}, v_i = v) \Rightarrow (d = v) \quad \text{e} \quad d \in D \tag{1}$$

   Se tutti i processi corretti propongono lo stesso valore v, allora il valore di decisione d deve essere v. Inoltre, il valore deciso d deve essere uno dei valori inizialmente proposti D.

2. **Accordo (Agreement)**:

$$\forall p_i, p_j \in \text{corretti}, \quad d_i = d_j \tag{2}$$

   Tutti i processi corretti devono convergere sulla stessa decisione d.

3. **Integrità (Integrity)**:

$$p_i \text{ decide } d_i \Rightarrow d_i \in D \text{ e } p_i \text{ decide al più una volta} \tag{3}$$

   Ogni processo decide al massimo una volta, e il valore deciso è un valore che è stato proposto.

2.2 Consenso in Sistemi Sincroni con Crash Failure

Nei sistemi sincroni, il consenso è **garantito** se il numero di fallimenti è tollerabile.

- **Ipotesi**: Sistema Sincrono; N processi; al più F fallimenti per **crash** ($F < N$).
- **Primitiva**: Si utilizza **B-Multicast**, che garantisce la ricezione da parte dei destinatari corretti se il mittente non fallisce.

Algoritmo e Dimostrazione 1 (Consenso in Sistemi Sincroni (Crash Failure))**.** *Per tollerare F fallimenti per crash, sono necessari $K = F + 1$ round (cicli) di comunicazione.*

Passaggi e Meccanismo: *L'algoritmo si basa sullo scambio iterativo del set di valori proposti, garantendo che l'informazione del fallimento o del valore di un nodo si propaghi attraverso tutta la rete.*

1. ***Inizializzazione (Round 0)**: Ogni processo P_i ha un vettore di valori proposti $V_i^{(0)}$.*
2. ***Iterazioni** ($k = 0$ a $F - 1$):*
   - *P_i invia, tramite $B\_multicast$, l'insieme dei valori $\Delta V_i^{(k)}$ che ha ricevuto e non ancora trasmesso nei round precedenti.*
   - *P_i riceve i messaggi e aggiorna il suo set di valori noti:*

$$V_i^{(k+1)} = V_i^{(k)} \cup \{\text{nuovi valori ricevuti nel round } k\}$$

3. ***Decisione Finale (Dopo $F + 1$ round)**: Al termine dell'ultimo round ($k = F$), tutti i processi corretti P_i avranno ricevuto l'insieme completo di valori noti nonostante i fallimenti. Essi scelgono un valore basandosi su una funzione di scelta deterministica (es. massimo o minimo):*

$$\text{Decidi: } d_i = \max(V_i^{(F+1)})$$

***Perché $F + 1$ round?** La convergenza è garantita perché il processo che fallisce nel round r ha inviato il suo valore nei round precedenti. Se il processo più lento o quello che fallisce per ultimo è cruciale, sono necessari F round per propagare l'informazione della sua assenza e un ulteriore round per raggiungere l'accordo finale tra i restanti. Dopo F round, l'informazione sui fallimenti dei primi F nodi si è propagata.*

3 Varianti e Limiti del Consenso

3.1 Problema dei Generali Bizantini (BG)

Il problema BG è il consenso in presenza di fallimenti **arbitrari (bizantini)**.

- **Scenario**: Un Comandante invia un ordine a $N - 1$ Luogotenenti.
- **Sfida**: I nodi bizantini possono inviare ordini diversi (es. Attacca e Ritirati) ai Luogotenenti corretti.

Teorema 1 (Soluzione BG in Sistemi Sincroni)**.** *Per risolvere il problema dei Generali Bizantini in un sistema sincrono, è necessario che il numero totale di processi N sia **strettamente maggiore di tre volte il numero dei fallimenti bizantini** F:*

$$N > 3F$$

In altre parole, il sistema può tollerare al massimo $F = \lfloor (N - 1)/3 \rfloor$ fallimenti bizantini.

Questo è un requisito molto più stringente del caso Crash, perché la malizia richiede ridondanza molto maggiore.

3.2 Consistenza Interattiva (IC)

Il problema della Consistenza Interattiva è una generalizzazione di C e BG, in cui l'obiettivo non è convergere su un singolo valore, ma su un **vettore di valori proposti** (uno per ogni processo).

- **Relazione con C e BG**: Una soluzione per IC implica una soluzione per BG, e una soluzione per BG è sufficiente per risolvere C.

$$\text{IC} \Rightarrow \text{BG} \Rightarrow \text{C}$$

3.3 Teorema di Impossibilità di FLP

Teorema 2 (Teorema di Impossibilità (FLP), 1985)**.** *Non esiste un algoritmo di consenso* ***deterministico*** *in un sistema distribuito* ***asincrono*** *che possa tollerare il fallimento di* ***anche un solo processo per crash****.*

Spiegazione Discorsiva: In un sistema asincrono, è impossibile distinguere un processo molto lento (ma corretto) da un processo che ha avuto un crash. Un algoritmo deterministico, per poter garantire la Validità e l'Accordo, si troverebbe in un vicolo cieco in cui non può decidere se aspettare il processo "lento" (violando il Progresso/Terminazione) o se procedere senza di esso (rischiando di violare l'Accordo se il processo non ha fallito).

3.4 Paxos e la Sicurezza Asincrona

Il protocollo di **Paxos** (Lamport) è la soluzione più nota per aggirare l'impossibilità di FLP nei sistemi asincroni.

- **Focus**: Paxos garantisce la proprietà di **Safety** (Validità e Accordo) anche in presenza di fallimenti asincroni.
- **Compromesso**: Non può garantire la proprietà di **Liveness** (Progresso o Terminazione) in presenza di fallimenti o di condizioni avverse (come una lunga sequenza di proposte concorrenti, sebbene sia raro). Questo è il prezzo da pagare per operare in un ambiente asincrono.

4 Consenso Sicuro e Autenticazione

Nelle reti reali, gli attacchi possono: far divergere la rete, invertire lo stato o impedire il consenso. Per mitigare i fallimenti bizantini e gli attacchi, si introduce la sicurezza.

4.1 Tecnica di Autenticazione Basata sull'Identità (Identity-Based Cryptography - IBC)

L'autenticazione tramite crittografia è un miglioramento fondamentale per gli algoritmi di consenso in contesti non affidabili.

Algoritmo e Dimostrazione 2 (Schema di Autenticazione con IBC)**.** *Si ipotizza l'uso della crittografia basata sull'identità per garantire l'autenticità dei messaggi e prevenire la manipolazione da parte di nodi bizantini.*

1. ***Firma del Messaggio****: Quando un nodo mittente P_m vuole inviare un valore v agli altri, esso:*
   - *Firma il messaggio con la sua* ***chiave privata*** *(SK_m).*
   - *Crittografa il messaggio utilizzando l'****identificativo del destinatario*** *(ID_r), che funge da chiave pubblica implicita.*
2. ***Invio****: Il mittente P_m invia il messaggio criptato e firmato.*
3. ***Verifica del Ricevente****: Il nodo ricevente P_r riceve il messaggio e compie due passaggi cruciali:*
   - ***Decrittografia****: Utilizza la sua* ***chiave privata*** *(SK_r) per decifrare il messaggio.*

- ***Verifica della Firma**: Utilizza l'**identificativo del mittente** (ID_m), che funge da chiave pubblica, per verificarne l'autenticità.*

4. ***Ingresso nel Consenso**: Se la verifica della firma ha successo, l'autenticità del mittente è garantita e il messaggio può entrare nella procedura di aggiornamento del consenso.*

Questo meccanismo previene che un nodo bizantino si spacci per un altro o che alteri un messaggio in transito.

5 Consenso Sicuro e Autenticazione

Il consenso sicuro affronta il problema degli **attacchi** in ambienti non fidati.

5.1 Effetti degli Attacchi sul Consenso

Gli attacchi (spesso di natura bizantina) agli algoritmi di consenso possono provocare i seguenti fenomeni:

- **Divergenza della Rete**: I nodi convergono su valori diversi, violando l'Agreement.
- **Inversione di Stato**: Alterazione dello stato corretto del sistema.
- **Convergenza al Valore Iniettato**: La rete converge sul valore imposto dall'aggressore.
- **Impedire il Consenso**: La rete non raggiunge mai la decisione finale, violando la Liveness.

5.2 Tecniche di Rilevamento e Autenticazione

5.2.1 Soluzione Basata sulla Devianza

Una prima soluzione per mitigare gli aggressori è calcolare la devianza di ogni nodo dal valore di consenso della maggioranza, e definire affidabili i nodi con una deviazione minima.

$$\text{Affidabilità di } P_i = f(\text{Devianza}(P_i, \text{Consenso}_{\text{Maggioranza}}))$$

Problema: Questo metodo può rimuovere anche utenti non aggressori se colludono, o diventare inefficace quando gli aggressori colludono fra loro, mimando il comportamento della maggioranza.

5.2.2 Crittografia Basata sull'Identità (IBC)

L'autenticazione tramite crittografia è la tecnica fondamentale per prevenire che un nodo bizantino alteri o spacci messaggi.

Algoritmo e Dimostrazione 3 (Schema di Autenticazione con IBC)**.** *L'autenticità si garantisce attraverso la firma del mittente e la crittografia per il destinatario.*

1. ***Firma e Crittografia**: Il mittente P_m firma con la sua chiave privata (SK_m) e crittografa usando l'identificativo del destinatario (ID_r) come chiave pubblica.*
2. ***Verifica del Ricevente** (P_r):*
   - ***Decrittografia**: Usa SK_r.*
   - ***Verifica Firma**: Usa ID_m (chiave pubblica del mittente) per validare l'autenticità.*

   Se la verifica ha successo, il messaggio può entrare nella procedura di consenso, sapendo che proviene sicuramente da P_m.

5.3 Consenso Basato sul Leader (Paxos)

Il protocollo di **Paxos** opera in sistemi asincroni aggirando il Teorema FLP.

- **Ruoli Chiave**: Paxos definisce tre ruoli che possono essere assunti da qualsiasi nodo:
  1. **Proposer**: Propone un valore e guida l'algoritmo.
  2. **Acceptor**: Vota e memorizza il valore accettato.
  3. **Learner**: Decide il valore finale.
- **Consenso Dinamico**: Si basa sull'elezione di un Leader (Proposer) che coordina le fasi del consenso. In caso di fallimento del Proposer, ne viene eletto uno nuovo.
- **Compromesso**: Paxos garantisce la **Safety** (Validità e Accordo) sempre, ma non può garantire la **Liveness** (Progresso/Terminazione) se i Proposer continuano a fallire o a competere tra loro continuamente.

6 Comunicazione di Gruppo e Rilevamento di Fallimenti (Lezione 2)

6.0.1 Gruppi Aperti e Chiusi

Una comunicazione di gruppo può essere classificata in base a chi è autorizzato a inviare messaggi al gruppo:

- **Gruppo Chiuso:** Solo i membri del gruppo possono inviare messaggi in multicast al gruppo stesso. Qualsiasi comunicazione dall'esterno viene bloccata.
- **Gruppo Aperto:** Anche entità non membre del gruppo possono inviare messaggi in multicast a tutti i membri del gruppo.

6.0.2 Strategie per la Consegna Affidabile

L'obiettivo della comunicazione di gruppo è assicurare che i messaggi vengano scambiati correttamente anche in presenza di fallimenti di canali e/o processi. Le strategie principali sono:

- **Error Masking:** Si basa sull'uso della ridondanza per mascherare gli errori di omissione.
  - **Ridondanza Spaziale:** I processi sono connessi tramite percorsi multipli. Per mascherare k omissioni, sono necessari $k + 1$ collegamenti distinti. Il problema principale è la possibile ricezione di messaggi duplicati, che devono essere scartati dal ricevente.
  - **Ridondanza Temporale:** Il messaggio viene inviato più volte sullo stesso canale. Per mascherare k omissioni, il messaggio deve essere inviato $k + 1$ volte. Anche in questo caso, è necessario gestire i duplicati.
- **Error Detection and Recovery:** Si basa sull'uso di messaggi di riscontro (acknowledgements) e timeout per rilevare e recuperare i messaggi persi.
  - **Positive Acknowledgements (ACK):** Il ricevente invia un ACK per ogni messaggio ricevuto. Il mittente ritrasmette il messaggio se non riceve l'ACK entro un timeout predefinito. Questo schema è rapido nell'individuare fallimenti anche con traffico sporadico.

– **Negative Acknowledgements (NACK):** Il ricevente notifica al mittente solo la mancata ricezione di un messaggio atteso. Questo minimizza il traffico di rete ma richiede meccanismi aggiuntivi, come la numerazione sequenziale dei messaggi, per permettere al ricevente di identificare le perdite.

6.0.3 Reliable Multicast

Un protocollo di *Reliable Multicast* (R-Multicast) deve soddisfare le seguenti proprietà per i processi corretti (che non falliscono):

- **Integrity:** Un processo corretto p riceve un messaggio m al più una volta, solo se p appartiene al gruppo destinatario e m è stato inviato dal mittente specificato.
- **Validity:** Se un processo corretto invia in multicast un messaggio m, prima o poi riceverà m esso stesso.
- **Agreement:** Se un processo corretto riceve un messaggio m, allora tutti i processi corretti nel gruppo prima o poi riceveranno m.

Una versione più forte di questa proprietà è la **Uniform Agreement**, che si applica indipendentemente dal fatto che il processo che riceve il messaggio sia corretto o meno: se un processo (corretto o meno) riceve m, allora tutti i processi corretti nel gruppo prima o poi riceveranno m.

6.0.4 Ordinamenti dei Messaggi in Multicast

Per garantire la consistenza, è spesso necessario che i messaggi inviati a un gruppo vengano consegnati a tutti i membri secondo un ordine specifico.

Ordinamento FIFO (First-In, First-Out). Se un processo corretto invia in multicast un messaggio m e successivamente un messaggio m', allora ogni processo corretto che riceve m' deve aver già ricevuto m. Questo ordinamento preserva l'ordine di invio dei messaggi provenienti da un singolo mittente.

Ordinamento Causale. Questo modello estende l'ordinamento FIFO per rispettare la potenziale relazione di causalità tra messaggi inviati da processi diversi. Si basa sulla relazione *happened-before* ($\rightarrow$). Formalmente:

- Se un processo p invia m e un processo p' riceve m prima di inviare m', allora nessun processo corretto riceve m' senza aver prima ricevuto m.
- In altre parole, se $multicast(g, m) \rightarrow multicast(g, m')$, ogni processo corretto che riceve m' deve aver già ricevuto m.

L'ordinamento causale implica quello FIFO.

Ordinamento Totale. Se un processo corretto riceve un messaggio m e successivamente un messaggio m', allora ogni altro processo corretto che riceve entrambi i messaggi deve ricevere m prima di m'.

- L'ordinamento di consegna è lo stesso per tutti i processi membri del gruppo.
- Tuttavia, questo ordine può essere arbitrario e non implica necessariamente l'ordinamento FIFO o quello causale. Ad esempio, un messaggio inviato prima potrebbe essere consegnato dopo, purché l'ordine sia lo stesso per tutti.

6.0.5 Implementazione dell'Ordinamento Totale

Metodo con Sequenziatore. Un approccio si basa su un processo designato come **sequenziatore**, che ha il compito di assegnare un numero di sequenza univoco e progressivo a ogni messaggio inviato al gruppo.

1. Un processo mittente invia un messaggio m sia al gruppo g sia al sequenziatore.
2. Il sequenziatore riceve i messaggi, assegna loro un numero di sequenza progressivo (s_g) e comunica questo numero al gruppo tramite un messaggio di ordinamento via B-multicast.
3. Ogni processo membro del gruppo riceve sia il messaggio originale m sia il messaggio di ordinamento dal sequenziatore. Mantiene i messaggi in una coda di attesa (*hold-back queue*) e li consegna all'applicazione solo quando il loro numero di sequenza corrisponde al prossimo numero atteso.

Metodo Distribuito (Algoritmo ISIS). Un approccio alternativo, valido per gruppi aperti o chiusi, prevede che i processi concordino collettivamente sui numeri di sequenza. L'algoritmo si articola in tre fasi:

1. Un processo invia un messaggio m a tutti i membri del gruppo.
2. Ogni ricevente propone un numero di sequenza per quel messaggio, tipicamente basato sul numero di sequenza più alto che ha visto finora, e lo invia solo al mittente originale. Mantiene il messaggio in una coda locale ordinata in base al numero proposto.
3. Il mittente raccoglie tutte le proposte di sequenza, seleziona il valore più alto come numero di sequenza finale ("agreed") e lo comunica a tutto il gruppo. I riceventi aggiornano il numero di sequenza del messaggio nella loro coda e lo consegnano quando diventa il messaggio in testa alla coda.

6.0.6 Tecniche di Propagazione

Oltre al B-Multicast, esistono tecniche di diffusione (broadcast) meno strutturate:

- **Flooding (Inondazione)**:
  1. Il mittente invia il messaggio a tutti i suoi vicini.
  2. Ogni nodo ricevente (se è la prima volta che vede il messaggio) lo inoltra a tutti i suoi vicini, eccetto quello da cui l'ha ricevuto.

  Vantaggio: Raggiunge rapidamente tutti i nodi. **Svantaggio:** Enorme overhead di messaggi e rischio di cicli.

- **Gossiping (Diffusione a Pettegolezzo)**:
  1. Un nodo mittente P_i invia il messaggio solo a un **sottoinsieme casuale** dei suoi vicini (fan-out).
  2. I nodi riceventi, con una certa probabilità, inoltrano il messaggio ad altri vicini scelti casualmente.

  Vantaggio: Scalabile, basso overhead, robusto ai fallimenti. **Svantaggio:** L'affidabilità (la garanzia di consegna) è **probabilistica**, non deterministica.

6.1 Rilevatori di Fallimenti (Failure Detectors)

I Failure Detectors sono componenti che tentano di identificare i processi falliti. Si classificano per Completezza (identificazione) e Precisione (accuratezza).

[Classificazione dei Failure Detectors - Riveduta]

6.1.1 Classificazione Formale (Chandra e Toueg)

I Failure Detector (FD) vengono classificati in base a due metriche fondamentali:

- **Completezza (Completeness):** La capacità del detector di rilevare effettivamente i fallimenti.
  - *Strong Completeness:* Prima o poi, ogni processo fallito è permanentemente sospettato da **ogni** processo corretto.
  - *Weak Completeness:* Prima o poi, ogni processo fallito è permanentemente sospettato da **almeno un** processo corretto.
- **Accuratezza (Accuracy):** La capacità del detector di non sospettare erroneamente processi corretti (evitare falsi positivi).
  - *Strong Accuracy:* Nessun processo corretto è mai sospettato da alcun altro processo corretto.
  - *Weak Accuracy:* Esiste almeno un processo corretto che non è mai sospettato da alcun altro processo corretto.
  - *Eventual Strong Accuracy:* Esiste un istante di tempo dopo il quale nessun processo corretto è più sospettato da alcun processo corretto.
  - *Eventual Weak Accuracy:* Esiste un istante di tempo dopo il quale esiste almeno un processo corretto che non è più sospettato da alcun processo corretto.

Combinando questi livelli si ottengono otto classi di failure detector. Il più debole in grado di risolvere il consenso in un sistema asincrono è l'**Eventually Weak Failure Detector** ($\Diamond W$).

6.1.2 Implementazione Pratica dei Failure Detector

L'implementazione di un FD inaffidabile in un sistema asincrono si basa tipicamente su meccanismi di *heartbeat*:

1. Ogni processo p_j invia periodicamente (ogni T secondi) un messaggio "I-am-alive" a tutti gli altri processi.
2. Ogni processo p_i monitora gli altri. Se p_i non riceve un messaggio da p_j entro un timeout $T + \Delta$ (dove Δ è una stima del tempo massimo di trasmissione), sospetta che p_j sia fallito e lo marca come `Suspected`.
3. Se successivamente p_i riceve un messaggio da p_j, ritira il sospetto e lo marca come `Unsuspected`.

La scelta di T e Δ è un compromesso: valori brevi aumentano il traffico e i falsi positivi; valori lunghi ritardano il rilevamento. Una soluzione pratica prevede l'uso di valori adattativi.

Esistono tre principali architetture per il monitoraggio:

- **Centralizzata:** Un singolo processo monitora tutti gli altri. Semplice ma introduce un Single Point of Failure (SPOF).

- **Ad Anello:** Ogni processo monitora il suo successore in un anello logico. Non ha SPOF ma è vulnerabile a fallimenti multipli consecutivi.
- **Distribuita (Gossip-based):** Ogni processo mantiene una lista di membri con i relativi contatori di heartbeat e la propaga periodicamente tramite gossiping. Questo approccio è molto robusto, scalabile e tollerante ai fallimenti. Il tempo di propagazione di un aggiornamento è $O(\log N)$.

7 Replicazione e Modelli di Consistenza

La replicazione dei dati nei sistemi distribuiti è fondamentale per migliorare l'affidabilità e le prestazioni. Tuttavia, introduce una sfida cruciale: mantenere la consistenza tra le diverse copie. Un **modello di consistenza** è un contratto tra i processi e l'archivio dati: se i processi rispettano determinate regole, l'archivio si comporterà come previsto. L'obiettivo è definire quali tra le innumerevoli possibili sequenze di esecuzione (interleaving) delle operazioni di lettura e scrittura sono valide.

7.1 Modelli di Consistenza Basati su Architettura (Data-Centric)

Questi modelli mirano a fornire una vista consistente dell'archivio dati a livello di sistema. L'idea è simulare il comportamento di un sistema centralizzato, anche se con diversi gradi di "rilassamento" delle garanzie per migliorare le prestazioni.

7.1.1 Consistenza Stretta (Strict Consistency)

È il modello più forte e intuitivo.

- **Definizione**: Qualsiasi operazione di lettura su un dato x restituisce il valore corrispondente al risultato dell'operazione di scrittura più recente su x, secondo un tempo globale assoluto.
- **Caratteristiche**: Le scritture sono viste come istantanee e atomiche da tutti i processi. Richiede l'esistenza di un orologio globale, rendendolo un modello ideale ma praticamente irrealizzabile nei sistemi distribuiti reali a causa dell'assenza di tale orologio.

7.1.2 Linearizzabilità (Linearizability)

È un leggero indebolimento della consistenza stretta, ma il più forte modello realizzabile in pratica.

- **Definizione**: Il risultato di qualsiasi esecuzione è lo stesso che si otterrebbe se tutte le operazioni di tutti i processi fossero eseguite in un qualche ordine sequenziale. Questo ordine deve essere compatibile con l'ordine temporale reale delle operazioni non sovrapposte. Una lettura deve sempre restituire il valore della scrittura più recente, indipendentemente da chi l'ha invocata[cite: 720].
- **Implementazione**: Si può simulare un tempo globale tramite clock sincronizzati (es. con NTP). È un modello costoso perché richiede un accordo (consenso) sull'ordine totale delle operazioni.

7.1.3 Consistenza Sequenziale (Sequential Consistency)

Questo modello rilassa ulteriormente il vincolo temporale.

- **Definizione (Lamport)**: Il risultato di ogni esecuzione è lo stesso come se le operazioni di tutti i processi fossero eseguite in un qualche ordine sequenziale, e le operazioni di ogni singolo processo appaiono in questa sequenza nell'ordine specificato dal suo programma.
- **Differenza con la Linearizzabilità**: L'ordine globale delle operazioni non deve necessariamente rispettare l'ordine temporale reale. L'unica cosa che conta è che tutti i processi vedano la stessa sequenza di operazioni. Questo dà al sistema più libertà, rendendolo più performante.

7.1.4 Consistenza Causale (Causal Consistency)

Questo modello indebolisce la consistenza sequenziale distinguendo tra operazioni causalmente correlate e operazioni concorrenti.

- **Definizione**: Tutte le scritture che sono potenzialmente correlate causalmente devono essere viste da tutti i processi nello stesso ordine. Le scritture concorrenti possono essere viste in ordini diversi su macchine diverse.
- **Relazione di Causalità**: Una scrittura W1 causa una scrittura W2 se W2 è avvenuta dopo che il processo che la esegue ha letto il valore scritto da W1. Se due scritture non sono legate da questa relazione (direttamente o transitivamente), sono considerate concorrenti.

Se il processo P1 scrive 'W(x)a', e successivamente il processo P2 legge 'R(x)a' e poi scrive 'W(x)b', allora 'W(x)a' e 'W(x)b' sono causalmente correlate. Tutti i processi devono vedere 'a' prima di 'b'. Se un terzo processo P3 scrive 'W(y)c' in modo indipendente, questa scrittura è concorrente con le altre e può essere vista in ordine diverso.

7.2 Teorema CAP (Brewer's Theorem)

Il Teorema CAP stabilisce un limite fondamentale per i sistemi distribuiti.

(Teorema CAP). È impossibile per un sistema di archiviazione dati distribuito fornire simultaneamente più di due delle seguenti tre garanzie:

1. **Consistenza (Consistency)**: Ogni lettura riceve la scrittura più recente o un errore (tutti i nodi vedono gli stessi dati nello stesso momento).
2. **Disponibilità (Availability)**: Ogni richiesta riceve una risposta (non un errore), senza garanzia che contenga la scrittura più recente.
3. **Tolleranza al Partizionamento (Partition Tolerance)**: Il sistema continua a funzionare nonostante un numero arbitrario di messaggi vengano persi (o ritardati) dalla rete tra i nodi.

Poiché le partizioni di rete sono un dato di fatto nei sistemi distribuiti su larga scala, la scelta reale è tra consistenza e disponibilità.

- **Sistemi CP (Consistency + Partition Tolerance)**: In caso di partizione, il sistema sceglie di sacrificare la disponibilità per garantire la consistenza. Bloccherà le richieste che non possono essere soddisfatte in modo consistente.

- **Sistemi AP (Availability + Partition Tolerance)**: In caso di partizione, il sistema sceglie di sacrificare la consistenza (forte) per rimanere disponibile. Risponderà alle richieste, anche se potenzialmente con dati obsoleti. Questo porta al modello di **Eventual Consistency**.

7.3 Filosofie di Design: ACID vs BASE

Queste due filosofie rappresentano gli estremi dello spettro consistenza-disponibilità.

- **ACID (Atomicity, Consistency, Isolation, Durability)**: È l'approccio tradizionale dei database relazionali, che privilegia la consistenza forte (sistema CP). È un approccio pessimistico, meno scalabile per volumi di dati enormi.
- **BASE (Basically Available, Soft state, Eventual consistency)**: È l'approccio dei sistemi NoSQL moderni, che privilegia l'alta disponibilità (sistema AP).
  - **Basically Available**: Il sistema è quasi sempre disponibile.
  - **Soft State**: Lo stato del sistema può cambiare nel tempo anche senza input, a causa della convergenza finale.
  - **Eventual Consistency**: Se non vengono effettuati nuovi aggiornamenti, tutte le repliche alla fine convergeranno sullo stesso stato.

7.4 Risoluzione dei Conflitti (Reconciliation)

Nei sistemi a consistenza finale, possono verificarsi conflitti quando repliche diverse vengono aggiornate in modo concorrente durante una partizione. Le strategie per risolverli includono:

- **Last Writer Wins (LWW)**: Si sceglie la versione con il timestamp più recente. Per gestire la causalità in modo robusto, si usano spesso orologi vettoriali invece di timestamp fisici.
- **Risoluzione a livello Applicativo**: Il datastore presenta le versioni divergenti all'applicazione, che implementa una logica personalizzata per fonderle in un'unica versione corretta.

8 Crittografia e Comunicazione Sicura

La crittografia è la disciplina che si occupa di proteggere le informazioni, garantendo proprietà fondamentali come la **confidenzialità**, l'**integrità** e la **disponibilità** dei dati[cite: 15, 16].
[cite$_s$*tart*]*Lasuaapplicazionecrucialeincontestiinsicuri*, *comelamemorizzazionedidatisudispositivivulne*

8.1 Principi di Crittografia

(Processo Crittografico). Un processo crittografico trasforma un **testo in chiaro** (plaintext) in un **testo cifrato** (ciphertext) tramite un'operazione di **cifratura** (encryption). Il processo inverso è la **decifratura** (decryption).

Il processo si basa su due elementi:

- **Algoritmo**: La sequenza di passi (pubblica e nota a tutti) per cifrare e decifrare.
- **Chiave (Key)**: Un'informazione segreta che guida l'algoritmo. La robustezza di un sistema crittografico moderno risiede interamente nella segretezza e nella lunghezza della chiave. Rompere un cifrario senza la chiave dovrebbe richiedere un tempo computazionalmente proibitivo.

8.2 Crittografia Simmetrica e Asimmetrica

8.2.1 Crittografia a Chiave Simmetrica

Utilizza la **stessa chiave** sia per la cifratura che per la decifratura.

- **Vantaggi**: È estremamente veloce ed efficiente.
- **Svantaggi**: La gestione e la distribuzione sicura della chiave segreta sono complesse. Le parti devono scambiarsi la chiave tramite un canale sicuro preesistente, perché se un attaccante la intercetta, la comunicazione è compromessa.

8.2.2 Crittografia a Chiave Asimmetrica (a Chiave Pubblica)

Utilizza una **coppia di chiavi** matematicamente correlate: una **chiave pubblica** (condivisibile) e una **chiave privata** (segreta).

- **Confidenzialità**: Per inviare un messaggio segreto a Bob, Alice lo cifra con la chiave pubblica di Bob. Solo Bob potrà decifrarlo con la sua chiave privata.
- **Autenticazione e Non Ripudio**: Se Alice cifra (firma) un messaggio con la sua chiave privata, chiunque può usare la sua chiave pubblica per verificare che il messaggio provenga da lei e non sia stato alterato. Poiché solo lei possiede la chiave privata, non può negare di averlo generato.
- **Svantaggi**: È computazionalmente molto più lenta della crittografia simmetrica.

Nei sistemi pratici (es. protocolli SSL/TLS), si usa un approccio ibrido: la crittografia asimmetrica viene usata per scambiare in modo sicuro una chiave di sessione, che viene poi utilizzata con un algoritmo simmetrico più veloce per la comunicazione vera e propria.

8.3 Firme Elettroniche e Digitali

Una firma elettronica è l'equivalente digitale di una firma autografa e serve a dare valore legale a un documento informatico. Il quadro normativo di riferimento è il regolamento europeo **eIDAS** e, in Italia, il **Codice dell'Amministrazione Digitale (CAD)**.

8.3.1 Tipologie di Firme Elettroniche

- **Firma Elettronica Semplice (FES)**: Dati in forma elettronica connessi logicamente a un'identità. Esempi: firma scannerizzata, PIN del bancomat, login con username/password. Il suo valore probatorio è molto debole e liberamente valutabile dal giudice.
- **Firma Elettronica Avanzata (FEA)**: È collegata unicamente al firmatario, idonea a identificarlo e creata con mezzi sotto il suo controllo esclusivo. Esempio: firma grafometrica su tablet in banca. Ha valore legale equiparato alla firma autografa.
- **Firma Elettronica Qualificata (FEQ)**: È una FEA creata con un dispositivo qualificato e basata su un certificato qualificato. È la forma più forte. In caso di contenzioso, l'onere della prova si inverte: è il firmatario a dover dimostrare che la firma è falsa.

8.3.2 La Firma Digitale

La firma digitale è una particolare implementazione tecnologica di una **Firma Elettronica Qualificata** che si basa sulla crittografia asimmetrica e su un sistema di certificati.
(Processo di Firma Digitale).

1. Si calcola l'impronta (digest) del documento tramite una funzione di hash.
2. Al digest si concatena una **marcatura temporale**, che certifica data e ora della firma.
3. Il risultato viene cifrato con la chiave privata del firmatario.
4. Il documento originale e la firma vengono inseriti in una **busta crittografica** (es. PAdES per PDF, CAdES per file .p7m).

8.4 Steganografia vs Crittografia

Entrambe le tecniche mirano a proteggere le informazioni, ma con approcci diversi.

- **Crittografia**: Occulta il *contenuto* del messaggio, ma non la sua esistenza. Un messaggio cifrato è visibile, anche se incomprensibile.
- **Steganografia**: Occulta l'*esistenza stessa* del messaggio. Il dato segreto viene nascosto all'interno di un altro dato apparentemente innocuo (cover data), come un'immagine o un file audio. Una tecnica comune è modificare i bit meno significativi (LSB) dei pixel di un'immagine per codificare il messaggio nascosto.

8.5 Crittografia Classica e Criptoanalisi

La crittografia classica si basava su algoritmi semplici di sostituzione e trasposizione.

- **Cifrario di Cesare**: Un cifrario a sostituzione monoalfabetica dove ogni lettera viene spostata di un numero fisso di posizioni nell'alfabeto.
- **Criptoanalisi**: È l'arte di "rompere" i cifrari. Il metodo più antico ed efficace contro i cifrari a sostituzione semplice è l'**analisi delle frequenze**. Poiché ogni lingua ha una distribuzione di frequenza caratteristica per le sue lettere (es. 'e' in italiano), si può analizzare la frequenza dei simboli nel testo cifrato per dedurre le sostituzioni e decifrare il messaggio.