Replication & Consistency in Distributed Systems

Distributed Systems

Andrea Omicini

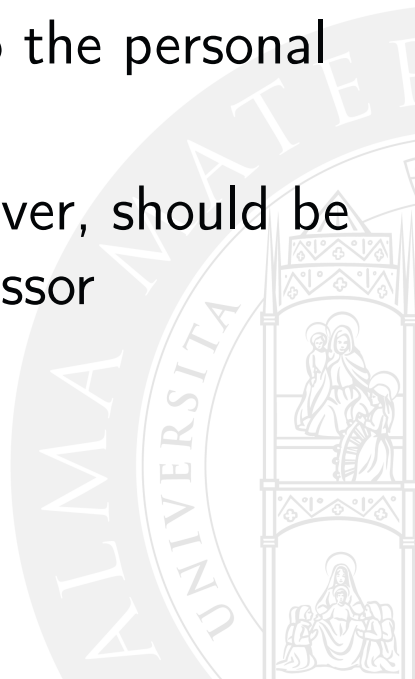
mailto:andrea.omicini@unibo.it
Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna

Academic Year 2025/2026

Next in Line...

Disclaimer

- the following slides borrow a great deal from the slides coming with one book used in this course [Tanenbaum and van Steen, 2017]
  - obviously along with other material from other books and articles
- material from those slides (including pictures) has been re-used in the following, and integrated with new material according to the personal view of the professor
- every problem or mistake contained in these slides, however, should be attributed to the sole responsibility of this course's professor



Next in Line...

Reasons for Replication I

Assicurare che la stessa informazione sia presente su più nodi/server.

Replication of *data*

- increasing the *reliability* of systems
- improving *performance*
- *scaling*
  - in numbers
  - in geographical area

Tolleranza ai Guasti (Fault Tolerance): se un nodo fallisce, gli altri continuano a servire le richieste (disponibilità). Questo è il motivo principale per la replica dei dati.

Latenza Ridotta (Latency): Le richieste possono essere servite dal nodo più vicino o meno carico, riducendo il tempo di risposta per l'utente.

- Scalabilità: La capacità di gestire un carico di lavoro maggiore distribuendo le richieste su più repliche (Load Balancing).
- Scalabilità Orizzontale (Load Scaling): Gestire un aumento del volume di richieste/utenti replicando i server.
- Prossimità Geografica (Geo-Scaling): Migliorare la latenza posizionando le repliche vicino agli utenti.

Reasons for Replication II

Failover & High Availability (Disponibilità (Availability) Post-Guasto): Questo descrive il Failover, un meccanismo per passare rapidamente a una replica sana. Sottolinea che la replica garantisce Alta Disponibilità.

Data replication for *reliability*

- if a file system has been replicated, system operation may continue after one replica crashes by simply switching to one of the other replicas
- maintaining multiple copies allows for better protection against corrupted data
  - esempio:
  - e.g., if we have three copies of a file, and perform every read and write operation on each copy, we could hide a single failing write operation by considering as correct the value returned by at least two copies

Voting/Quorum e Mascheramento dei Guasti: Questo è un esempio perfetto di un protocollo basato su Quorum (es. lettura da 2 su 3) o di un sistema di Tolleranza ai Guasti Bizantini (BFT). L'idea è che la maggioranza vince, mascherando l'errore di un singolo nodo.

Integrità dei Dati: Si distingue dalla semplice disponibilità. Non si tratta solo di crash (guasti totali), ma anche di errori silenti o corruzione dei dati.

Reasons for Replication III

Size: Un numero crescente di processi/client deve accedere ai dati di un singolo server (gestione del carico di lavoro). Soluzione: Replicare il server e distribuire il carico (Load Balancing) tra le repliche. Ideale per carichi intensivi di lettura (Read-Heavy).

Data replication for *performance*

- replication for performance is important when a distributed system needs to scale in terms of *(i)* size or *(ii)* the geographical area covered

L'obiettivo è aumentare il throughput (numero di operazioni al secondo).

- e.g., scaling regarding size occurs when an increasing number of processes needs to access data managed by a single server
  - performance can be improved by *(i)* replicating the server, and *(ii)* subsequently dividing workload among the processes accessing the data

L'obiettivo è diminuire il tempo di risposta per i client remoti.

- e.g., when scaling over a geographical area, time to access data decreases by placing a copy of data near to the process using them

→ performance as *perceived* by that process increases

! keeping all replicas up to date may consume more network bandwidth

? do we have a benefit overall?

Geo Area: Gli utenti sono distribuiti geograficamente (necessità di prossimità). Soluzione: Posizionare copie dei dati vicino ai processi che le usano.

Reasons for Replication IV

I termini Reliability e Fault Tolerance sono molto vicini, ma in alcuni contesti si distinguono: FT è la capacità di continuare a operare nonostante i guasti; Reliability è la probabilità di operare senza guasti per un certo periodo. In un sistema distribuito, la replica migliora entrambi.

General *benefits* of replication in distributed systems

- at a first glance, benefits of (data) replication
  - reliability
  - fault tolerance
  - accessibility
  - performance
  - scalability

Gestione dei guasti e alta disponibilità.

I dati sono più facili da raggiungere, anche se un nodo è offline (sinonimo di disponibilità).

Gestione del carico e riduzione della latenza.

look unquestionable

? so, should we just replicate (data, processes, whatever), and be content with it?

Issues of Replication |

Costi Fissi (CapEx): Risorse Computazionali: Acquisto di nuovi nodi server e infrastruttura di rete.
Costi Variabili (OpEx): Bandwidth: Il costo per sincronizzare gli aggiornamenti tra i nodi replicati. Questo costo scala con la frequenza delle scritture.
Costi di Gestione: Personale Tecnico: Necessità di sistemi di monitoraggio e orchestratori complessi per gestire il failover e la sincronizzazione.

Problems in distributed systems

- costs
  - computational resources
  - bandwidth
- consistency

Costs of replication

- replicating data in a distributed system first of all means that some (new?) machines, elsewhere located, should be used (bought?), and possibly placed near to the distributed processes in need of accessing data, so as to host the replicas

  Replica = Costo Hardware (Acquisizione) e Costo Logistico (Posizionamento Geografico).
- more management costs (physical spaces, technical personnel, . . .), possible acquisition costs

  Copre i Costi Operativi (OpEx) come manutenzione, spazio fisico, e personale tecnico.

Issues of Replication II

Replication requires computation and bandwidth

- replicating data is effective if the copies are *consistent* with the original
  - or, more generally, if all the replicas are consistent with each other
- which requires *computation* within replicas, and *communication* among replicas
  - so, *bandwidth*, too

la sincronizzazione richiede risorse di calcolo (per processare e ordinare gli aggiornamenti) e banda di rete (per trasmettere gli aggiornamenti).

Main problem of replication: consistency

una scrittura singola (o update) crea divergenza.

- whenever a copy is modified it becomes different from the others

Propagazione e Sincronizzazione.

- modifications have to be carried out on all copies to ensure consistency

! when and how modifications should be carried out determines the *cost* of replication

Il quando (immediato vs. differito) e il come (sincrono vs. asincrono, votazione, ecc.) definiscono il trade-off.

Sincronizzazione Immediata-Massima consistenza, dati sempre aggiornati.-Bassa performance, alta latenza di scrittura.
Sincronizzazione Differita-Alta performance, bassa latenza di scrittura.-Gli utenti potrebbero leggere dati vecchi o diversi.

Issues of Replication III

Example: Improving access to Web pages

- if no special measures are taken, fetching a page from a remote Web server may sometimes even take seconds to complete
- to improve performance, Web browsers often locally store a copy of a previously fetched Web page—i.e., they *cache* a Web page
- if a user requires that page again, the browser can returns the local copy
- so that the user perceives an excellent access time (latenza eccellente)
- problem: what if the page has been modified on the server in the meantime, the cached copy is out-of-date, yet the user wants / needs the latest version of the page?

Issues of Replication IV

Forbidding caching (Vietare la cache):
- Consistenza Forte (Strong Consistency): I dati sono sempre freschi perché ogni accesso va alla fonte.
- Performance Povera: Si nega il vantaggio principale della replica (latenza ridotta), influenzando negativamente la performance percepita.

Improving access to Web pages: Possible solutions

Returning a *stale copy* (copia vecchia) to the user could be prevented

- by forbidding caching
  - thus badly affecting perceived performance
- by putting the server in charge of invalidating / updating cached copies
  - the server needs to keep track of all caches and send them messages, which is heavy load, leading to poor scalability

Server invalidating/updating cached copies:
- Consistenza Forte Gestita (Cache Coherence): Il server garantisce che le copie siano aggiornate (attraverso l'invalidazione).
- Scalabilità Povera: Il server/Master deve mantenere lo stato di tutte le repliche/cache. Questo è il problema del bottleneck del Master.

La consistenza forte e la scalabilità elevata sono mutuamente esclusive, cioè non possono verificarsi contemporaneamente (Teorema CAP).

Replication as a Scaling Technique I

l'utente percepisce la scarsa scalabilità come lentezza.

- scalability issues generally appear in the form of performance problems
- replication and caching for performance are widely applied as scaling techniques
- e.g., placing copies of data close to the processes using them can improve performance through reduction of access time, and thus solve scalability problems (Questo copre la latenza)
- trade-off: keeping copies up to date may require more network bandwidth Questo riconferma che la performance di lettura ottenuta è bilanciata dal costo di consistenza/sincronizzazione (consumo di banda).

Replication as a Scaling Technique II

N: Numero di accessi (letture) al secondo da parte del processo P. (Beneficio di performance/latenza).
M: Numero di aggiornamenti (scritture) al secondo alla replica (costo di consistenza/banda).
Condizione Critica: N≪M (Letture molto meno frequenti delle Scritture).

An example

- a process P accesses a local replica N times per second
- whereas the replica itself is updated M times per second
- with each update completely refreshing the previous version of the local replica

! if access-to-update ratio is very low ($N \ll M$), most updated versions of the local replica will never be accessed by P, thus making network communication for those versions useless

? is the extra-effort of placing a local replica worth the cost?

? or, should a different strategy for updating be applied?

Questo significa che, se i dati cambiano velocemente ma vengono letti raramente, la maggior parte della banda di rete spesa per sincronizzare la replica locale viene sprecata. La replica diventa un net loss in termini di efficienza.

Replication as a Scaling Technique III

Questo significa che un'operazione di scrittura non può considerarsi completata finché tutte le repliche non hanno confermato l'aggiornamento. Questo impone un ordine globale.

Another problem

- keeping multiple replicas *consistent* may itself be become the source of serious scalability problems
- intuitively, a collection of copies is consistent when the copies are always the same
- one (desired?) consequences that a read operation performed at any copy will always return the same result
- to this end, when an update operation is performed on one copy – any copy –, the update should be propagated to all copies before a subsequent operation takes place
- which is a lot of computation and communication
- ! by the way, that would be *synchronous replication*, and the result is somehow (informally) called *tight consistency*

Indica che la scrittura è bloccante per il client fino a quando tutte le repliche non hanno risposto. Questo garantisce la consistenza ma massimizza la latenza di scrittura.

le operazioni appaiono istantanee e in un ordine globale che rispetta l'ordine in tempo reale.

Replication as a Scaling Technique IV

Qui si ripresenta il trade-off: velocità vs. atomicità. L'aggiunta di latenza di rete e potenziali guasti rende questa operazione complessa.

Key idea and issues

- an update is performed at all copies as a single *atomic operation*, or *transaction*
  Questo è l'ideale della consistenza forte, dove una modifica su un nodo è vista come un evento unico e indivisibile su tutti i nodi.
- unfortunately, implementing atomicity involving many replicas is inherently difficult when operations are also required to be fast
- main problem: we need to synchronise *all* replicas
  requisito per ottenere la Consistenza Forte
- this means that *all replicas* need first of all to reach agreement on when exactly an update is to be performed locally

Questo "accordo su quando" ha due implicazioni cruciali:

  - e.g., all replicas may need to agree on the global ordering of (distributed) operations
    Questo è il cuore dei problemi di consistenza. Due operazioni concorrenti devono essere viste in un ordine univoco da tutte le repliche.
- global synchronisation take a lot of communication (time), and—is it (always) possible, or are we bumping into another set of impossibility theorems?

Ordine (Ordering): Le repliche devono concordare su un ordine globale unico delle operazioni, specialmente in presenza di scritture concorrenti.
Tempo (Timing/Atomicity): L'aggiornamento deve avvenire come un'unica operazione atomica percepita da tutti.

Replication as a Scaling Technique V

The replication and consistency dilemma

- Migliora il throughput e riduce la latenza in lettura (Performance/Scalabilità)
  on the one hand, scalability problems can be alleviated by applying replication and caching, leading to improved performance
- Aumenta la latenza di scrittura e introduce un overhead di banda (Consistenza)
  on the other hand, to keep all copies consistent generally requires global synchronisation, which is inherently costly in terms of performance

? is the cure worse than the disease?

- la scelta della "migliore" soluzione dipende dalle esigenze specifiche dell'applicazione
  as usual, there is no general answer to this problem, valid for every sort of distributed system—yet, some solution is quite often available

Replication as a Scaling Technique VI

Questo significa che le scritture non sono più bloccanti e possono essere completate localmente (o su un sottoinsieme di nodi) prima di essere propagate asincronamente.

Relaxing consistency constraints

- often, the only real solution is to *relax the consistency constraints*
- that is, by relaxing the requirement that updates need to be executed as atomic operations, we may avoid (instantaneous) global synchronisations
  - così thus gaining performance — L'operazione di scrittura diventa molto più veloce (bassa latenza di scrittura).
- however, this means that replicas may not always be the same everywhere — Questo introduce la possibilità di leggere dati obsoleti (stale data) o valori diversi da repliche diverse.

Dipende dalla semantica dell'applicazione e dalle conseguenze di un errore.

? is this generally acceptable, or, in which cases is it such?

? more precisely, when and to what extent (in che misura) consistency can be relaxed?

! this typically depends on the access and update patterns of replicated data, as well as on the purpose of data usage

La consistenza può essere rilassata quando:
Le letture superano di gran lunga le scritture
L'Obiettivo è l'Alta Disponibilità
Le operazioni possono essere eseguite in qualsiasi ordine senza cambiare il risultato finale

Next in Line...

The Problem of Consistency

Consistency models

Si rompe con l'idea di "consistenza perfetta" (che è troppo costosa) e si apre alla possibilità di compromesso.

- instead of a single notion of consistency, we may define different sorts of consistency, each one fitting different application scenarios
- so that engineers can fully explore the trade-off between the costs and the benefits of consistency
  - by relaxing consistency requirements according to the specific application scenario at hand
- this has lead to the definition of different consistency models...

scegliere il Modello di Consistenza che offre sufficienti garanzie per l'applicazione, senza pagare l'eccessivo overhead della sincronizzazione globale.

... which are then amenable of different implementations, based on different protocols, and whose features may affect the effectiveness of the model

realizzabili tramite

efficacia

Focus on...

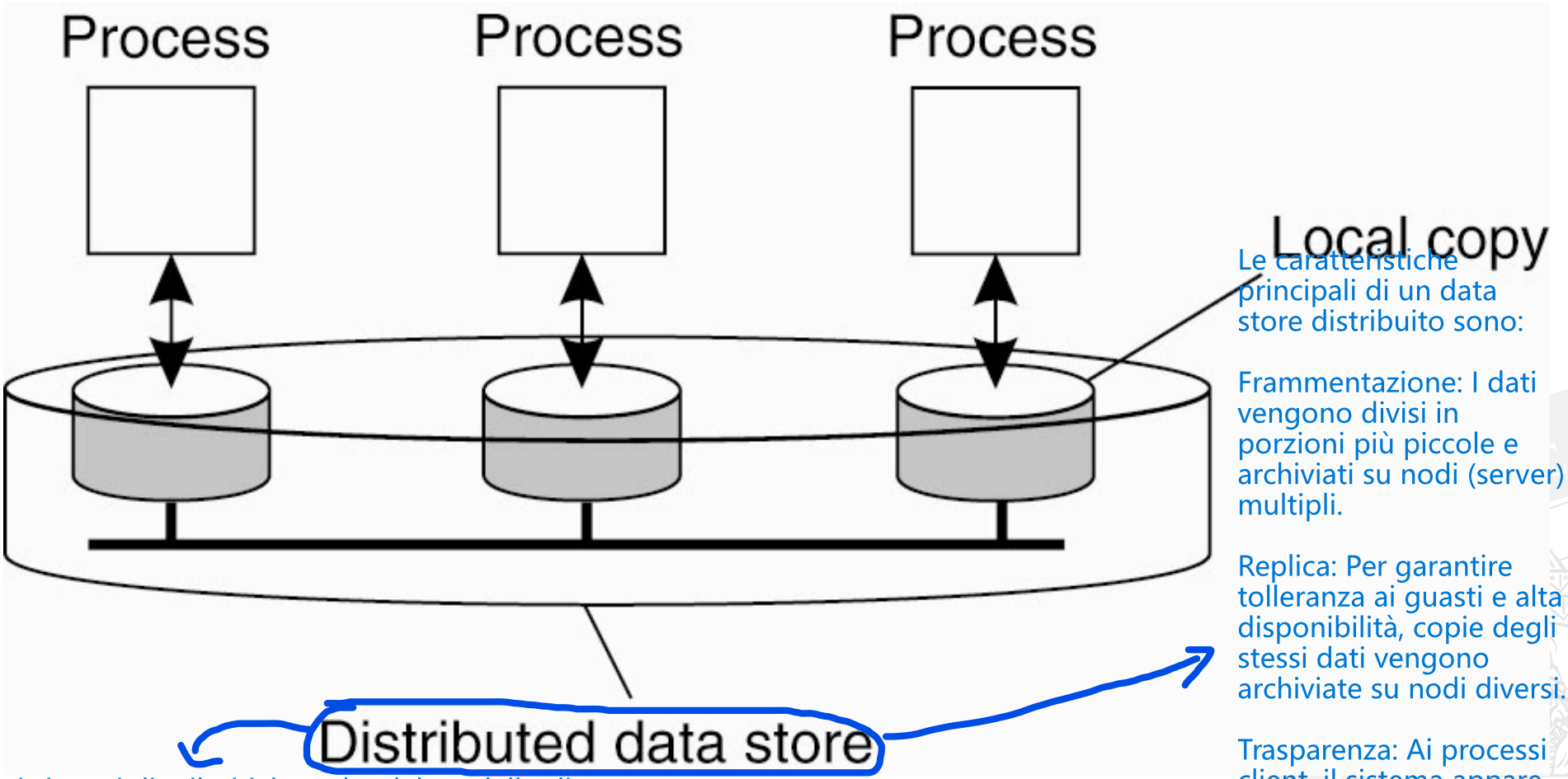
Conceptual Premise

Replicating data

- historically, the first things to be distributed are data
- so, the first problem to be addressed is how to ensure *consistency of data* across distributed copies...
- ... and the actions to be accounted for, are *operations over data*

Questo punto restringe il campo d'azione. Non ti concentrerai su processi o risorse generiche, ma sulle operazioni di lettura (read) e scrittura (write), che sono le operazioni fondamentali che un sistema deve garantire in modo coerente.

General Organisation of a Distributed Data Store



[Tanenbaum and van Steen, 2017]

Le caratteristiche principali di un data store distribuito sono:

Frammentazione: I dati vengono divisi in porzioni più piccole e archiviati su nodi (server) multipli.

Replica: Per garantire tolleranza ai guasti e alta disponibilità, copie degli stessi dati vengono archiviate su nodi diversi.

Trasparenza: Ai processi client, il sistema appare come una singola entità logica, nascondendo la complessità della distribuzione e della replica sottostante.

Nei sistemi distribuiti, la scelta del modello di consistenza (come la Consistenza Sequenziale o Causale) è il contratto che il data store offre ai processi per gestire l'inevitabile inconsistenza temporanea dovuta alla replica e alla latenza.

Consistency Model I

I modelli di consistenza non descrivono solo il comportamento del data store, ma stabiliscono anche le regole di comportamento che il client (process) deve seguire per ottenere la garanzia promessa (es. "se il client aspetta un certo tempo, vedrà l'aggiornamento").

Definition

A consistency model is essentially a contract *among the processes and the data stores*, ensuring the correctness of data given a set of rules that processes have to follow

Questo è il motivo per cui esistono modelli diversi. La "correttezza" per un'applicazione bancaria è diversa dalla "correttezza" per una bacheca di social media.

- of course, what is "correct" also depends on what processes expect
- which might also be problematic to define in absence of a global notion of system time

Questo è il problema centrale della sincronizzazione: non esiste un orologio universale (tempo globale) nei sistemi distribuiti. Senza questo orologio, è quasi impossibile determinare l'ordine assoluto di due eventi che avvengono su nodi diversi. I modelli di consistenza devono quindi operare su concetti di ordine parziale (causalità) o imporre un ordine totale virtuale.

Consistency Model II

La tradizionale idea di consistenza, tipica dei database centralizzati (ACID), si concentra sul dato: la garanzia che il dato sia sempre corretto e che tutte le repliche siano identiche in ogni momento. Nei sistemi distribuiti, questo ideale è troppo costoso. Il cambio di focus proposto significa: Non più: "Tutte le repliche sono identiche in ogni istante?", ma: "L'esperienza dell'utente (o processo) è logicamente coerente?"
Questo permette di introdurre Modelli di Consistenza Debole che migliorano la scalabilità, accettando che le repliche possano essere temporaneamente diverse, purché l'utente non sperimenti anomalie critiche per l'applicazione.

Note

- the above definition of consistency model shifts the focus from the *replicated data* to the *processes using data*
- so, focussing on the notion of consistency in the *use* of data, based on the specific application needs

Different sorts of contract

- since different application needs may lead to different useful notions of consistency, various *definitions of consistency model* are available
- we discuss some of them in the next slides

La varietà di modelli esiste perché il costo della consistenza forte (latenza di scrittura e bassa disponibilità in caso di partizionamento) è inaccettabile per molte applicazioni su larga scala.

Continuous Consistency

La Consistenza Continua non cerca di eliminare l'inconsistenza, ma di limitarla entro soglie definite.

Questo permette di pagare solo il costo di sincronizzazione necessario per rimanere entro quei limiti, non il costo di una sincronizzazione globale perfetta.

Goal

Imposing limits to *deviations* between replicas

Deviations

su cui l'inconsistenza può essere misurata e limitata

Level of consistency defined over three independent axes for deviation

- *numerical* deviations — absolute / relative
  - e.g., two copies a stock price should not deviate by more than $0,01

Quanto differiscono i valori tra le repliche.
Delta Assoluto (Δ): Differenza massima accettabile nel valore (es. 0.01$).

- *staleness* deviations
  - e.g., weather data should not be more than four hours stale

Quanto è vecchio (obsoleto) il dato letto rispetto all'ultima scrittura nota.
Tempo (τ): L'età massima accettabile della replica (es. 4 ore).

- *ordering* deviations
  - e.g., in a bulletin-board application, a maximum of six messages can be issued out-of-order

Quanti aggiornamenti possono essere ricevuti "fuori ordine" da una replica.
Numero di Aggiornamenti (k): Quanti aggiornamenti in sequenza (dal master) possono essere ignorati o riordinati localmente (es. 6 messaggi).

This defines the notion of continuous consistency

Inconsistency

Significa che è l'entità di dato considerata atomica per quanto riguarda l'applicazione delle regole di consistenza.
- Se modifichi un Conit: L'intera unità deve essere replicata e resa consistente. Non puoi avere una parte del Conit aggiornata su una replica e un'altra parte non aggiornata.
- Se due scritture avvengono sullo stesso Conit: Il sistema deve intervenire con un protocollo di sincronizzazione (es. consenso o risoluzione dei conflitti) per garantire che i nodi concordino sul valore finale.
- Se due scritture avvengono su Conit diversi: Possono avvenire in parallelo e in modo indipendente, migliorando la concorrenza.

How do we *measure deviation*?

- in the data-centric perspective, we use the traditional notion of unit of consistency — called *conit*

  È l'unità più piccola che viene considerata singolarmente consistente.

- the very nature of each data store either implicitly or explicitly suggests its conit

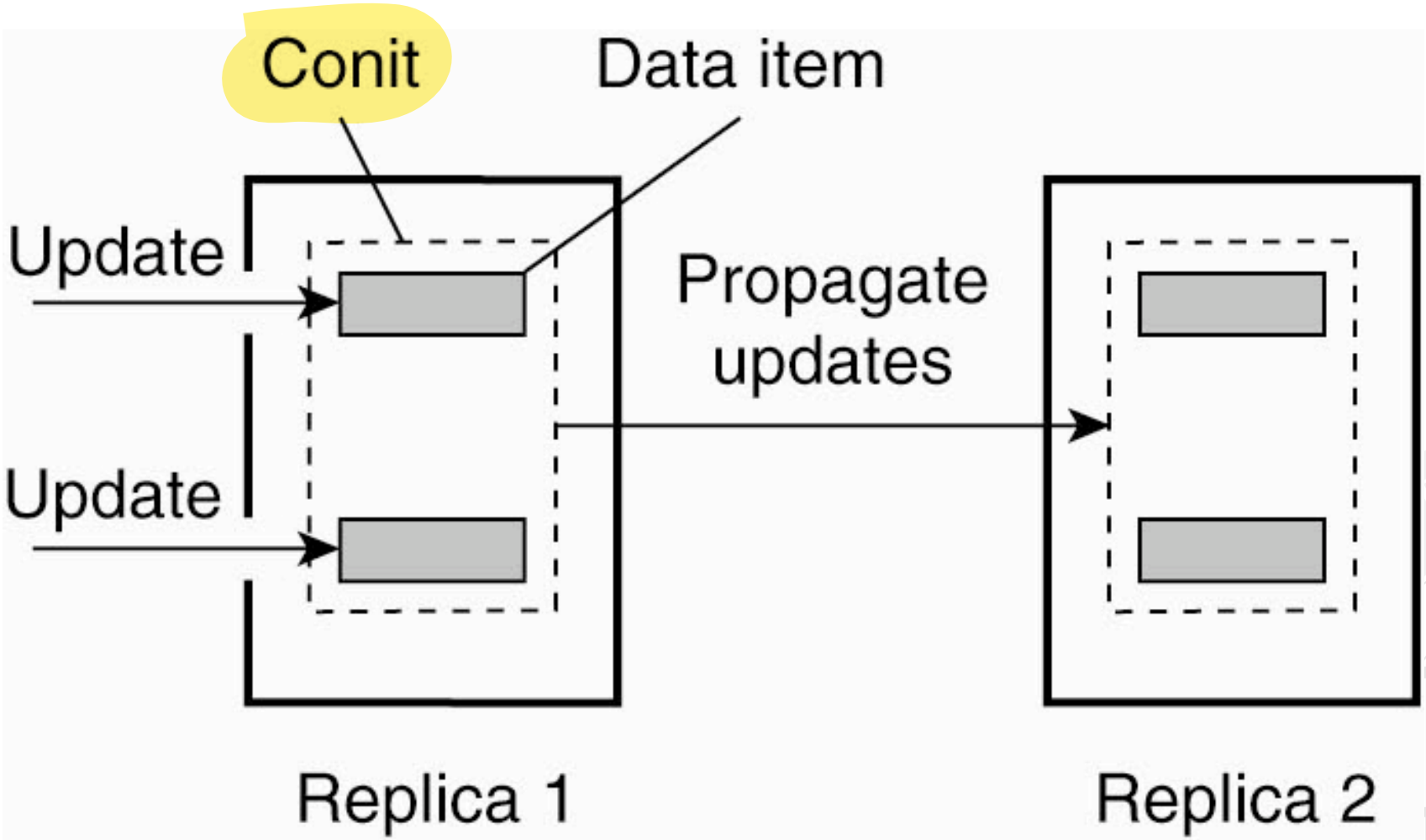
  Esempi di Conit:
  Database Relazionali: Una riga o una tabella.
  File System: Un blocco o un file.

- however, a consistency model (and replication) is defined around a suitably-designed notion of conit
- *deviation* is measured in terms of differences of conits

→ there is no an absolute measure for deviation: conit should be chosen carefully, depending on the resource and the problem at hand

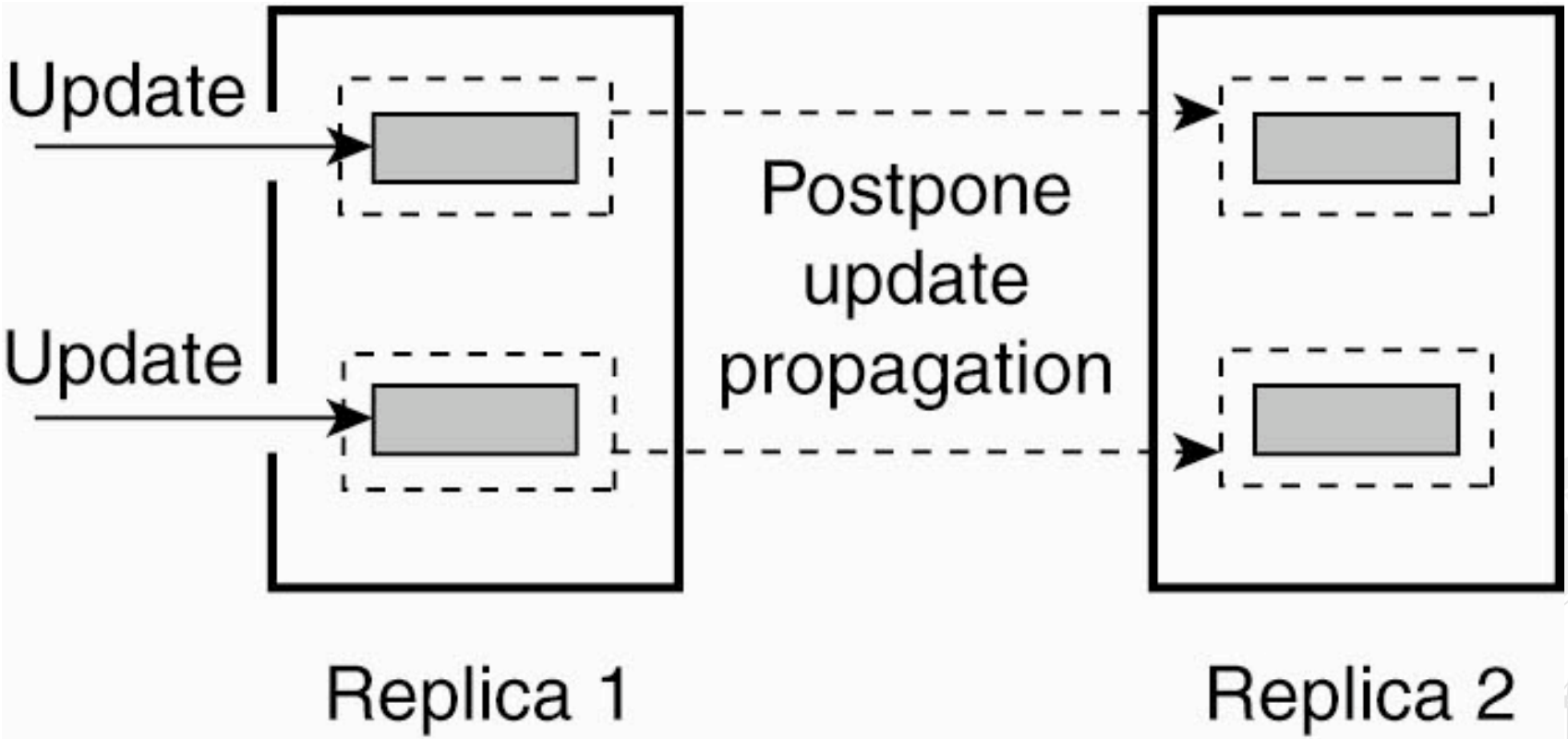
Questa scelta è un trade-off:
Conit Piccolo (Es. una singola riga): Permette alta concorrenza (meno probabilità di conflitti tra scritture) ma aumenta la complessità di gestione e il potenziale di inconsistenza parziale.
Conit Grande (Es. un intero documento): Semplifica la gestione della consistenza (è sufficiente aggiornare un'unica entità) ma riduce la concorrenza (due scritture che modificano punti diversi dello stesso documento devono attendere).

The Choice of Granularity of Conit I



Larger conit [Tanenbaum and van Steen, 2017]

The Choice of Granularity of Conit II



Smaller conit, minor need for propagation [Tanenbaum and van Steen, 2017]

Consistent Operations Ordering

La concorrenza (più scritture simultanee sullo stesso dato) è il catalizzatore che crea inconsistenza, poiché senza un coordinamento, repliche diverse potrebbero applicare le scritture in ordini diversi.

Main issue

- from parallel and concurrent environments, where several processes share resources, and have to access them simultaneously
- new models conceptually extending data-centric ones: when committing on a state for replicas, an *agreement* has to be reached among processes upon the global ordering of updates

Per mantenere la consistenza (soprattutto forte), il sistema deve imporre un ordine totale su tutte le scritture, anche se sono avvenute su nodi diversi in tempi leggermente diversi. Questo è l'obiettivo principale dei protocolli di consenso: far sì che i nodi concordino su quale operazione è successiva a un'altra, creando un registro logico comune.

Sequential Consistency

La Consistenza Sequenziale è un modello Data-Centric forte, cruciale perché garantisce che tutti i client vedano la storia del sistema nello stesso modo, risolvendo il problema dell'ordinamento globale

Questo è il requisito fondamentale: il sistema deve imporre un ordine totale univoco su tutte le operazioni di scrittura. Se il processo P1 vede l'ordine WA->WB, anche il processo P2 deve vedere WA->WB (e non WB->WA).

Main idea Ordine Totale Condiviso

- all update operations are seen by all processes in the same order

Definition

- a data store is sequentially consistent when the result of any execution is the same as if
  - the (read and write) operations by all processes on the data store were executed in some sequential order, and
  - the operations of each individual process appear in this sequence in the order specified by its program

Questo è il vincolo più importante: l'ordine stabilito non può riordinare le operazioni emesse da un singolo client. Se il client Pi ha eseguito Opi1 e poi Opi2, nella sequenza globale Opi1 deve apparire prima di Opi2.

Implica l'esistenza di una singola sequenza di operazioni (come se ci fosse un database centrale) che tutti i nodi replicano.

Causal Consistency

Questo modello indebolisce l'esigenza di un ordine totale globale, concentrandosi solo su ciò che è logicamente necessario ordinare: gli eventi legati da una relazione di causa-effetto.

La Consistenza Causale identifica le operazioni in cui un evento deve logicamente seguire un altro (es. una lettura che usa il valore di una scrittura precedente).

Se due operazioni non hanno una relazione di causa-effetto (es. due scritture eseguite da client diversi simultaneamente, senza che una legga l'altra), sono considerate concorrenti. Il sistema non è obbligato a ordinarle nello stesso modo per tutti i client.

Main idea

- weakening sequential consistency

  La Consistenza Sequenziale richiedeva un ordine totale per tutte le operazioni.
- based on the notion of cause/effect relation
- unrelated operations are *concurrent* ones
- ordering is limited to operations in cause/effect relation

Questo è il compromesso che garantisce la scalabilità: si paga il costo della sincronizzazione solo per le operazioni causali, permettendo alle operazioni concorrenti di procedere più liberamente e asincronicamente.

Definition

- a data store is causally consistent when all processes see write operations that are in a cause/effect relation in the same order

se l'operazione A ha causato l'operazione B, tutti i nodi devono vedere A prima di B.
Le operazioni non causali possono essere viste in ordine diverso dai vari client.

Focus on...

Switching Perspective

Sharing data in mobile computing scenario

- a client connects with different replicas over time
- differences between replicas should be made transparent
- no particular problems of simultaneous updates, here

Client-centric consistency models

- in essence, they ensure that whenever a client connects to a new replica, that replica is up to date according to the previous accesses of the client to the same data in the other replicas on different sites
- consistency is no longer referred directly to the resource – its nature, its dynamics, . . . –, but rather on the *view that each client has of the resource*

Eventual Consistency

Scenario

- large, distributed data store with almost no update conflicts
- typically, a single authority updating, with many processes simply reading
- the only conflict is *read-write conflict* where one process wants to update a data item while another concurrently attempts to read the same data
- examples: DNS changes, Web content

Issues

- non-updated data may be provided to readers
- in most cases, such an inconsistency might be acceptable to readers
- typically, if no update takes place for a while, gradually all replicas will become consistent
- this sort of consistency is called eventual consistency

Monotonic Reads

Definition

A data store is said to provide monotonic-read consistency if the following condition holds

- if a process reads the value of a data item x, any successive read operation on x by the process will always return that same value or a more recent value

Example

- distributed e-mail database

Monotonic Writes

Definition

A data store is said to provide monotonic-write consistency if the following condition holds

- a write operation by a process on a data item x is completed before any successive operation on x by the same process

Idea

- the order of updates is maintained over distributed replicas

Example

- software library under development

Read Your Writes

Definition

A data store is said to provide read-your-writes consistency if the following condition holds

- the effect of a write operation by a process on data item x will always be seen by a successive read operation on x by the same process

Idea

- avoid the "web page failed update" effect

Example

- password updating

Writes Follow Reads

Definition

A data store is said to provide writes-follow-reads consistency if the following condition holds

- a write operation by a process on data item x following a previous read operation on x by the same process is guaranteed to take place on the same or a more recent value of x that was read

Idea

- writes affect only up-to-date data items

Example

- comments to posts on FaceBook

Next in Line...

Replica Management

Supporting replication in a distributed system

- means deciding where, when and by whom replicas should be placed
- and which mechanisms should be adopted to keep replicas consistent

Two subproblems

- placing *replica servers*
- placing *content*
- ! not the same problem indeed

Service Replication

Replication does not mean replicating data — not merely

- *services* could be replicated as well in a distributed setting
  - for the same reasons of data stores
- this essentially means replicating functions, which may / may not insist on the same data store
- two layers for replications, with two consistency & replication models

Process Replication

Mobility & cloning for replication

- *processes* could also be replicated in a distributed mobile setting
- again, for the same reasons of data stores
- this might require cloning...
- ... but also higher-level mechanisms, like goal-passing

More on Replication I

Book "Replication. Theory and Practice"

- publisher's page of the book[Charron-Bost et al., 2010]
- just a minor example of what we mean with *"This is just a course on the basics of distributed systems"*
- many things have obviously to be left *out* of the course
- yet the point is: if you make the effort of *following* and *understanding* our flow here, at the end of the course you will be able to take that book and *read* it *proficiently*

More on Replication II

Other books on replication in general

- "Database Replication" [Kemme et al., 2010]
- "Replication Techniques in Distributed Systems" [Helal et al., 2002]
- a specific chapter in "Principles of Distributed Database Systems" [Özsu and Valduriez, 2020]

Other books on replication within specific DB frameworks

- MySQL [Bradford and Schneider, 2012, Schwartz et al., 2012]
- PostgreSQL [Böszörmenyi and Schönig, 2013]

Other books on replication within storage-agnostic tools

- Veeam Backup & Replication [Childerhose, 2022]

... Replication & Consistency for Fault Tolerance? I

- actually, so many available technologies around providing for that
- a fast scan of the Web leads to the table below

consistency type	*key mechanism / protocol*	*typical systems / tech*
strong	Raft, Paxos	etcd, Spanner, CockroachDB
eventual	Gossip, Merkle Trees	DynamoDB, Cassandra, Riak
causal	Vector Clocks, CRDTs	AntidoteDB, Orleans, Akka
tunable	Quorums (R/W)	Cassandra, Dynamo-style DBs
transactional	Consensus + TrueTime	Spanner, CockroachDB
log-based	ISR, CDC Streams	Kafka, Debezium, Aurora

... Replication & Consistency for Fault Tolerance? II

- from the table: a lot of things we need to see and explain before we can easily understand (almost) everything
  - e.g., logical time in distributed systems, group and coordination algorithms, ...
- meanwhile, a forthcoming class on Kubernetes by Mattia Matteini will give you some concrete examples

Next in Line...

Lessons Learnt

- replication is useful in distributed systems
- replication requires consistency
- consistency is not an absolute notion: different application needs mandates for different models of consistency
- we talk about data, yet nowadays we replicate whatever resource we need
  - e.g., cloning in mobile distributed systems *(forthcoming)*

Replication & Consistency in Distributed Systems

Distributed Systems

Andrea Omicini

mailto:andrea.omicini@unibo.it
Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna

Academic Year 2025/2026

References I

[Böszörmenyi and Schönig, 2013] Böszörmenyi, Z. and Schönig, H.-J. (2013).
PostgreSQL Replication.
Packt Publishing
https://www.packtpub.com/product/postgresql-replication/9781783550609

[Bradford and Schneider, 2012] Bradford, R. and Schneider, C. (2012).
Effective MySQL Replication Techniques in Depth.
Oracle Press / McGraw-Hill
https://www.mhprofessional.com/effective-mysql-replication-techniques-in-depth-9780071791861-usa

[Charron-Bost et al., 2010] Charron-Bost, B., Pedone, F., and Schiper, A., editors (2010).
Replication. Theory and Practice, volume 5959 of *Lecture Notes in Computer Science*.
Springer, Berlin, Heidelberg
DOI:10.1007/978-3-642-11294-2

[Childerhose, 2022] Childerhose, C. (2022).
Mastering Veeam Backup & Replication.
Packt Publishing, 2nd edition
https://www.packtpub.com/product/mastering-veeam-backup-replication-second-edition/9781803236810

[Helal et al., 2002] Helal, A. A., Heddaya, A. A., and Bhargava, B. B. (2002).
Replication Techniques in Distributed Systems.
Kluwer Academic Publishers
DOI:10.1007/b101825

References II

[Kemme et al., 2010] Kemme, B., Jiménez Peris, R., and Patiño-Martínez, M. (2010).
Database Replication.
Morgan & Claypool Publishers
DOI:10.1007/978-3-031-01839-8

[Özsu and Valduriez, 2020] Özsu, M. T. and Valduriez, T. (2020).
Principles of Distributed Database Systems.
Springer, 4th edition
DOI:10.1007/978-3-030-26253-2

[Schwartz et al., 2012] Schwartz, B., Zaitsev, P., and Tkachenko, V. (2012).
High Performance MySQL. Optimization, backups, and replication.
O'Reilly, 3rd edition
https://www.oreilly.com/library/view/high-performance-mysql/9781449332471/

[Tanenbaum and van Steen, 2017] Tanenbaum, A. S. and van Steen, M. (2017).
Distributed Systems. Principles and Paradigms.
Pearson Prentice Hall, 3rd edition
https://www.distributed-systems.net/index.php/books/ds3/