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

Replication of *data*

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

Reasons for Replication II

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
    - 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

Reasons for Replication III

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
- 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
- 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?

Reasons for Replication IV

General *benefits* of replication in distributed systems

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

  look unquestionable

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

Issues of Replication I

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
- more management costs (physical spaces, technical personnel, . . .), possible acquisition costs

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

Main problem of replication: consistency

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

→ 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

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
- 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

Improving access to Web pages: Possible solutions

Returning a *stale copy* 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

Replication as a Scaling Technique I

- 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
- trade-off: keeping copies up to date may require more network bandwidth

Replication as a Scaling Technique II

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?

Replication as a Scaling Technique III

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*

Replication as a Scaling Technique IV

Key idea and issues

- an update is performed at all copies as a single *atomic operation*, or *transaction*
- 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
- this means that *all replicas* need first of all to reach agreement on when exactly an update is to be performed locally
  - e.g., all replicas may need to agree on the global ordering of (distributed) operations
- global synchronisation take a lot of communication (time), and—is it (always) possible, or are we bumping into another set of impossibility theorems?

Replication as a Scaling Technique V

The replication and consistency dilemma

- on the one hand, scalability problems can be alleviated by applying replication and caching, leading to improved performance
- 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?

- 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

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
  - thus gaining performance
- however, this means that replicas may not always be the same everywhere

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

? more precisely, when and to what extent 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

Next in Line. . .

The Problem of Consistency

Consistency models

- 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 lea to the definition of different consistency models...

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

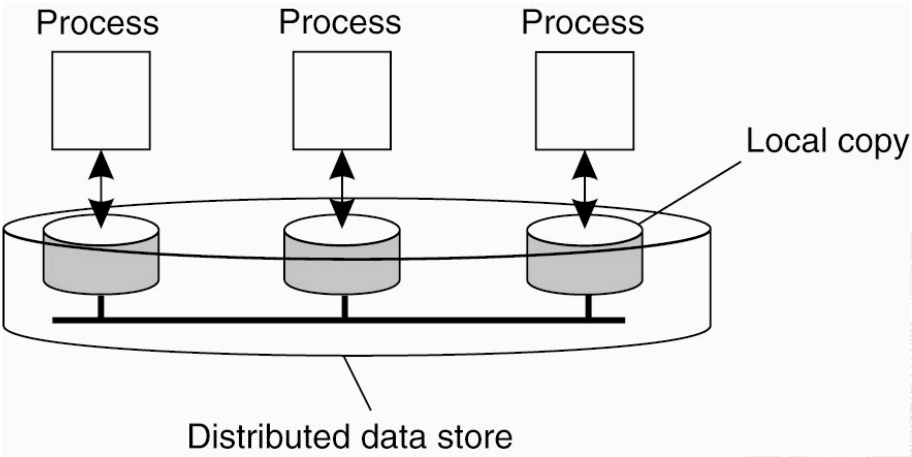
Focus on. . .

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*

General Organisation of a Distributed Data Store



[Tanenbaum and van Steen, 2017]

Consistency Model I

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

- 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

Consistency Model II

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

Continuous Consistency

Goal

Imposing limits to *deviations* between replicas

Deviations

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
- *staleness* deviations
  - e.g., weather data should not be more than four hours stale
- *ordering* deviations
  - e.g., in a bulletin-board application, a maximum of six messages can be issued out-of-order

This defines the notion of continuous consistency

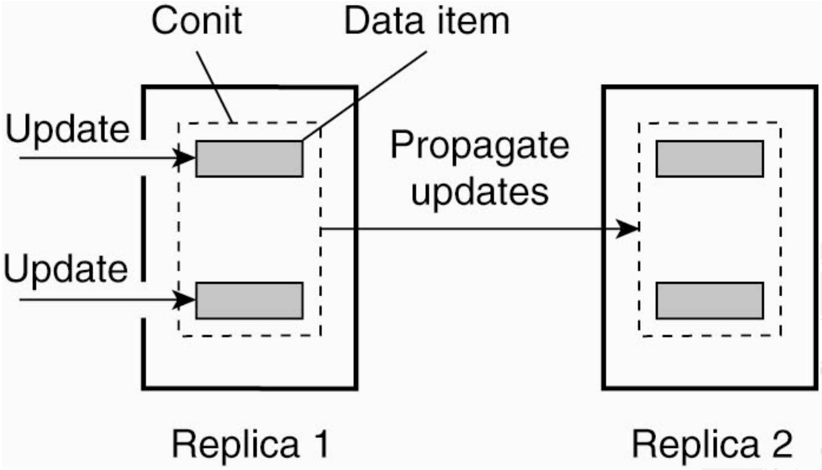
Inconsistency

How do we *measure deviation*?

- in the data-centric perspective, we use the traditional notion of unit of consistency — called *conit*
- the very nature of each data store either implicitly or explicitly suggests its conit
- however, a consistency model (and replication) is defined around a suitably-designed notion of conit
- *deviation* is measured in terms of differences of conits

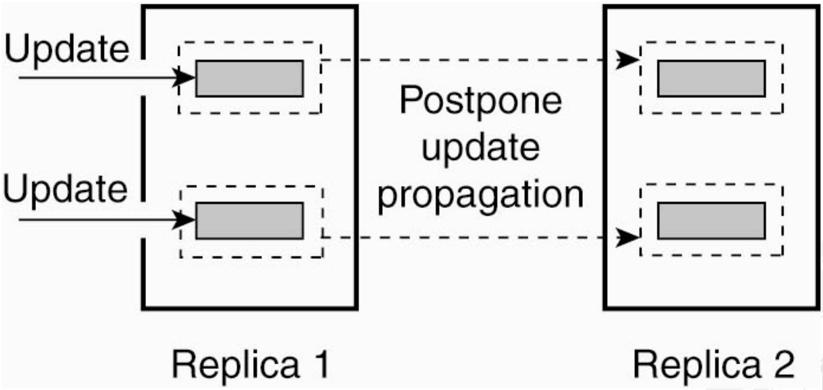
→ there is no an absolute measure for deviation: conit should be chosen carefully, depending on the resource and the problem at hand

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

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

Sequential Consistency

Main idea

- 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

Causal Consistency

Main idea

- weakening sequential consistency
- based on the notion of cause/effect relation
- unrelated operations are *concurrent* ones
- ordering is limited to operations in cause/effect relation

Definition

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

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/