Replication

Replication in Distributed Systems

Arianna Soriani
arianna.soriani@studio.unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna

Academic Year 2023/2024

Centralization no longer sufficient...

- Centralizing the resources is neither acceptable nor desirable. It introduces a potential performance bottleneck, as well as a single point of failure, the loss of which jeopardizes the entire range of functionality of the whole system.
- Also, distributed systems require fast access to resources they need, with low latency. Centralized resources have to deal with access management and response latency can be very high.
- The big advantage of centralized resources is the easiest and more reliable read and write operations over the resources.

Centralization no longer sufficient...

A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable. [Lamport, 1987]

Centralization no longer sufficient...

- Distributed computer would compose its resources so as to offer functionality, dependability, and performance, that far exceed those offered by a single isolated computer.
- A distributed computing system, being composed of a large number of computers and communication links, must almost always function with some part of it broken.

Why I

Why Replication

- Replication is the key to mobility, availability, and performance.
- It masks environmental failures (power, storms), hardware failures, operator errors, and even some software faults.
- Replication exploits locality of reference and read-intensive references to improve performance and scalability.
- Data is tipically read much more often than it is written.

Why II

Replication in distributed systems

Replication enables systems to remain distributed, while enhancing their availability and performance in the face of growing scale.

Data items, messages and processes can be replicated for:

- higher availability
- shorter response times
- enhanced recoverability
- hence reliability

Where I

Where to store replicas

- The decision is based on usage and storage costs.
- There are several variables to consider to perform replication:
  - e.g. location of major usage, system architecture, block size, etc.
  - they describe different replication techniques

Automating the decision of where to store what is one of the key problems in replication!

Where II

Data locality principle

When computations involves large set of data, it is cheaper (i.e. faster) to move code to data rather than data to code.

What

Replication of Data
Replication of Objects
Replication of Processes
Replication of Messages

Not only data!

Every component of a distributed system can be replicated!

Who

Everything!

Every distributed system can (and should) implement replication in their architecture.

- from small-scale distributed system $\rightarrow$to very big-scale distributed system

Look

The lots of advantages achieve with replication are exploitable also in distributed system with small number of nodes.

When

As a design feature!

- either at the application level
- or at the level of common system services
  - e.g. such as distributed databases, distributed file, hypermedia, or object-oriented systems, as well as inter-process communication systems and process management systems.
  - application-independent services.

Replication lies at the heart of the distributed computer architecture

Failures I

Failures arise from:

- *software bugs, human operator errors, performance overload, congestion, magnetic media failures, electronic component failures, or malicious subversion.*
- Also, *scheduled maintenance and environmental disasters* (e.g. fires, flood, earthquakes) shut down portions of distributed systems.

Failures II

Fault management

But...Distributed computing services *have to be maintained* in the presence of partial failures

- at the level of fault-isolation, or
- at the higher, more difficult, more expensive level of fault-tolerance

Fault-isolation

1. At the lower level
2. Requires only that the system contain any failures so that they do not spread (*isolation*)

SO...

The functionality of the failed part of the system is lost until the failure is repaired

If that is not acceptable, an operational component of the system take over the functionality of the failed pieces (fault-tolerance mechanism)

Fault-tolerance I

1. At the higher level
2. Requires:
   - reconfiguring the service to take advantage of new components that replace the failed ones
   - or designing the service so as to mask failures on-the-fly.

Fault-tolerance II

so... Replication!

Distributed redundancy represents a necessary underpinning for the design of distributed protocols (or algorithms) to:

- *mask* the effects of component failures
- *reconfigure* the system so as to stop relying on the failed component until it is repaired

Failures vs Errors

Failure

- A failure is enabled by the exercise of an error that leads to a faulty state
- A failure does not occur until the faulty state causes an *externally visible action* that departs from the set of acceptable behaviours.

Error

- Errors arise from *human mistakes in design, manufacturing or operation, or from physical damage to storage or processing devices*
- An error can lay dormant for a very long time before it ever causes a fault

Failures

Fault detection and prevention

When fault occurs, the subsystem may be able to correct its state (either by luck) or by fault-tolerant design, before any external ill-effects are released.

Software and Hardware tend to be designed so as to fail by shutting down-crashing when a fault is detected, before they produce erroneous externally visible actions.

Benign fault vs Malicious fault I

Benign fault

- Sites can be fail-stop when the processing and communication is terminated at the failed site *before* any external components are affected by the failure.
- Crash a process stops executing events.
- A link failure is when either if messages are no longer transmitted, or if they are dropped or excessively delayed.

Benign fault vs Malicious fault II

Malicious fault

- Malicious failures (arbitrary failures or byzantine failures)
- Sites fail and they begin to behave in malicious ways.
- Processes may lie
- A faulty component can exhibit arbitrarily malicious (so-called "Byzantine") behavior

System that tolerates Byzantine faults should then be able to handle everything!

Reliability x Availability = Dependability I

- For a service to be highly dependable, it must be both highly available
  - the probability that a request for the service will be accepted
- and highly reliable
  - conditional probability that the service will be carried through to successful completion

...but

As the number of sites involved in a computation increases $\rightarrow$
the likelihood decreases that the distributed system will deliver it functionality with acceptable availability and reliability

Reliability x Availability = Dependability II

High Availability

A system is highly available if *the fraction of its down-time is very small*, either because failures are rare, or because it can restart very quickly after a failure; also, if *denial of service request is rare*.

High Reliability

A system is highly reliable if, either *the system does not fail at all for a given lenght of time*, or it can *recover enough state information after a failure* for it to resume its operation as if it was not interrupted.

High Dependability

A system is highly dependable if it is both highly available and highly reliable.

Reliability

Definition

Reliability refers to the property of tolerating constituent component failures, for the longest time.

Perfect world

A system is perfectly reliable if it never fails

A world less perfect (...but more real!)

A system is reliable if it fails rarely and if it almost always recovers from component failures and design faults in such a way as to resume its activity without a perceptible interruption; so...

A system is reliable if it is able successfully to complete a service request, once it accepts it.

What is need

- A Fault-tolerance software component, with:
  - Failure detection protocols (i.e. surveillance protocols)
  - Recovery protocols
  - Failure adaptation and reconfiguration protocols

MTTF

- A simple and classical measure is MTTF (mean time to failure):
  - It is a measure of failure frequency that naturally lends itself to the exponential group of failure distributions
  - R(t), is the probability that the system functions properly in the interval [0,t]; it's the probability that the system's lifetime exceed t
  - R(t) can be determined from the system failure probability density function, f(x), and can be written as:

$$\boxed{R(t) = 1 - \int_0^t f(x)\,dx = 1 - F(t)}$$

- where F(t) is the cumulative probability distribution function

Recovery-enhanced reliability

- MTTF-based definition needs to be extended to includ the commitment to successfully completing a system operation.

Definition

A reliable system will take *all needed recovery actions* after a failure occurs, to *detect it, restore operation, and system states, and resume the processing of the temporarily interrupted tasks*.

- Unfortunately, not yet produced unified metrics to quantify recovery-enhanced reliability.

Availability

Definition

Availability refers to the accessibility of system services to the users.
A system is available if it is operational for an overwhelming fraction of the time.

Reliability vs Availability

- Unlike reliability, availability is *instantaneous*.
- A reliable system is not necessarily highly available.
- For frequently submitted (i.e. short-lived operations), availability is more significant than reliability.
- For infrequently requested (i.e. long-lived operations), reliability is more critical than availability.

Instantaneous availability

- The simplest measure of availability is the probability, A(t), that the system is operational at an arbitrary point in time t.
- It's the probability that either:
  - The system has been functioning properly in the interval [0,t], or
  - Tha last failed component has been repaired or redundantly replaced at time x, 0<x<t, and the repaired (redundant) component has been functioning properly sinche then (in [x,t]).
- It's easier to compute A(t) in systems that, in response to failures, use repairs, but not redundancy.

Limiting availability

$$\lim_{t \to \infty} A(t) = \frac{MTTF}{MTTF + MTTR}$$

- Availability can also be measured as:

$$A(t) = \frac{\sum_i u_i}{t}$$

- by observing the system over the interval [0,t] (i.e. mission time) and recording the periods of time, u_i
- the mission time is chosen equal to the utilization interval of the system

Dependability

Bayes' Law

$$\boxed{D_s = Pr[I_s(t) \cap T_s(\tau)] = A_s(t) * R_s(t, \tau)}$$

- τ is the duration of execution of a service s
- I_s(t) is the potential event that the system can successfully initiate s at time t
- $T_s(\tau)$ is the potential event that the system can terminate s successfully when its duration is τ.

System transparency

High Availability

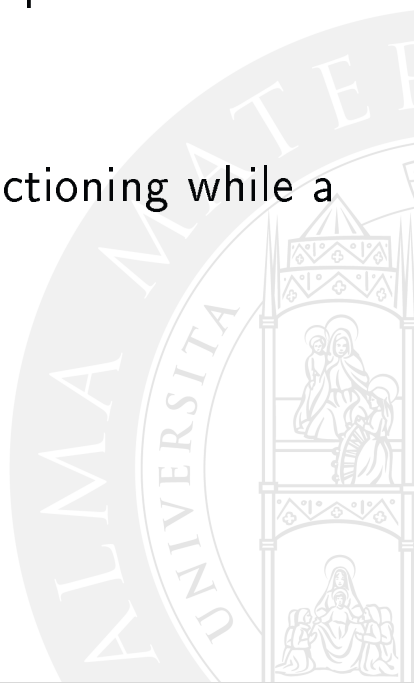
It's the most important feature that a distributed system must have to:

- mask component failure
- reconfigure the system so as to rely on different set of resources

- So... we must immediately discard the obvious option of composing distributed systems from *ultra-available* and *ultra-reliable*.
- As the size of the distributed system grows, its components' availability and reliability would also have to increase with it
  - violating a basic prerequisite for scalability in distributed systems: that its local characteristics be indipendent of its global size and structure.
- Therefore, the distributed system designer needs to instill mechanism that combat the effects of failures,*into the system architecture itself*

Strategies

- Strategies for dealing with component failures so as to achieve system dependability includes:
  - Failure detection, containment, repair (or restart), recovery
    - directly enhance the dependability of the individual component
  - Masking, and, Reconfiguration after a failure
    - Fault-tolerance methods
    - they are designed to enable the system to continue functioning while a failure is still unrepaired



Replication-based strategies

Replication techniques

Replication techniques mainly address the two fault-tolerance activities of *masking failures*, and of *reconfiguring* the system in response.

Benefits

- Fault-tolerance
- Failure detectability (e.g. mirroring)
- Recoverability

Replication lies at the heart of any fault-tolerant computer architecture

Replication-based strategies

Replication has the capacity to improve performance by bringing the aggregate computing power of all the replica sites, to bear on a single load category.

- e.g. each read operation can select the replica site from which to read, to minimize relevant costs, such as communication distance.

Limits and Costs

...but it is not perfect!

- Synchronizing and reconciling changes is actually very subtle.
- Replication of data, process, or messages, risks a number of ill consequences that can arise in the absence of failures.
- Unless carefully managed, replication jeopardizes consistency, reduces total system throughput, renders deadlock more likely, and weakness security.

Undesirable effects

Failures are generally indistinguishable from mere slowness

Limits and Costs

Consequences

Replication, be it of data, processes, or messages, provide fault-tolerance at the cost of duplicating effort

- by having the replicas carry out repetitive work, such as performing every data update multiple times, once at ever copy.
- high communication cost

That implies:

- storage space overhead
  - some coding-theoretic replication protocols (e.g. IDA) can reduce the storage cost of replication, in terms of both size and aggregate bandwidth
- overhead associated with consistency control across the replicas (with deadlock)

Limits and Costs

Raab's conclusion

With respect to limits on availability enhancement, the availability of update operations, in a replicated system, can be at most $\sqrt{A}$. [Raab, 1992]

- where A is the update availability of a single copy
- as measured by a certain metric, called *site availability*

Unfortunately, there has not been much other work in this important area.

DDBS

Definition

A distributed database system (DDBS) is a collection of data items scattered over multiple, networked, failure-independent sites, potentially used by many concurrent users.

Transparency

The database system manages this collection so as to render distribution, failures, and concurrency transparent to the users

- their reads (queries) and writes (updates) execute in a manner indistinguishable from reads and writes that execute on a *single-user*, reliable, *centralized* database.

Expensive and guaranteed in practice only to an extent consistent with *acceptable* performance

Transaction

Definition

A transaction consists of a set of operation executions performed during a run of a program

ACID

Transactions must possess ACID properties

- Atomicity: a transaction is performed in its entirely or not at all;
- Consistency: a transaction must take the database from one consistent state to another;
- Isolation: a transaction is isolated from ongoing update activities;
- Durability: a transaction update applied to the database has to be permanently installed.

Only consistency falls on the shoulder of the applcation programmer. The remaining three must be ensured by the DDBS.

Serialization

IF:

- the initial state of the database is *consistent* and
- each transaction is designed to preserve the database consistency (if executed in isolation)

THEN:

- an execution is equivalent to a *serial* one (with no transaction in inconsistent database).

Definition

An execution is *serializable* if it produces the same output and has the same effect on the database as some serial execution of the same transaction.

Concurrency Control Protocol

Definition

Concurrency Control Protocols are used to restrict concurrent transaction executions in a centralized and distributed database system only to executions that are serializable.

- pessimistic protocols: prevent inconsistencies by disallowing potentially non-serializable executions and by ensuring that the effects of a committed transaction need not be reversed or annulled.
- optimistic protocols: permit non-serializable executions to occur during a validation phase before the effects of the transactions are made visible

Atomicity Control Protocol

Definition

Atomicity control protocols ensure that each transaction is atomic.

Procedure:

- A *coordinator site* initiate a transaction
- read and write operations are executed by forwarding to the sites containing the relevant data items (i.e. *participants*).

But in a partitioned network, no atomic commitment protocol can guarantee the execution of a transaction to its termination as long as a network partition exists; failure necessitate of *recovery protocols*.

Read One Write All

The most obvious protocols are those that keep multiple copies that must all be updated, and of which any one can be read.

Simple ROWA protocol

This algorithm translate a *read* operation on data item d, into one read operation on any a single copy and a *write* operation into n write, one at each copy (n is the number of copies or sites).

- The concurrency controller at each site synchronizes access to copies
- Execution is equivalent to a serial execution (one update, will update all copies or none at all).

Simple ROWA protocol

So, a transaction that reads any copy of d, reads the most recent value, which is the one written by the last transaction that updated all copies.

Pro

The obvious advantages is its *simplicity* and its *ability to process reads despite site or communication failures*, so long as at least one site remains up and reachable.

Con

But, in the event of even one site being down or unreachable, the protocol would have to block all write operations until the failure is repaired.

Read One Write All Available Protocol

ROWA-A

A transaction is no longer required to ensure updates on all copies of a data item, but only on all the available copies.
Failed site are not allowed to become available again until they recover by copying the current value of the data item from an available copy.

Pro

This avoids the delay incurred by udate transactions when some sites in the system fail (ROWA-A can tolerate n-1 site failures).

Cons

ROWA-A does not tolerate network partitioning, nor communication failures.

ROWA-A Basic alghoritm

1. w[d] in a transaction T are sent to all the sites holding copies
2. if a site s is down, there will be no response from it and T's coordinator will timeout
3. is a site s is operational then it responds indicating whether w[d_s] was rejected or processed

The protocol is sufficient, if there are no risk that the coordinator may make a mistake regarding the status of participant.

Helps

Before committing the transaction, its coordinator initiates a two-step validation protocol:

- Missing writes validations: determines if all copies that caused missing writes are still unavailable.
- Access Validation: determines if all copies that it read from, or wrote into are still available.

Static Assignment

Cons

Static assignment of copies to sites has a number of disadvantages.

- e.g. repeated update attempts at a site which has been down for a long time is a waste of resources.

Solutions of High Communication Costs

- Dynamic creation and removal copies at new sites is not possible.
- But, the problem could be mitigated by buffering messages to the same site into a single message or by combining the communication involved in *access validation* with that of the *vote-req phase* of the atomic commitment protocol.

Primary Copy ROWA Protocol

Definition

- A specific copy is designated as primary copy
- The remaining copies are backups

How it works

- A *write operation* is carried out at the primary copy and all operational backups
- while, a *read operation* is executed only at a primary copy

When the primary fails, a backup, chosen via a fixed line of succession or by an election protocol, takes its place as the new primary.

Primary Copy ROWA Protocol

Cons

Replace primary copies failed, requires that the failure of the primary be *detectable* and *distinguishable* from failure to communicate with it (i.e. network partition).

- It's difficult to achieve in practice using commonly available network
- Some solution mechanism exists (e.g. combining with a voting-based protocol in which the primary votes its preparedness based on whether or not all of its backups agree [Oki and Liskov, 1988])

Pro

Even this method appears overly simple, it's one of most widely implemented replication techniques.

True Copy Token ROWA Protocol

Definition

Each data item d has always either one *exclusive* token associated with it or a set of *shared* tokens.

How it works

- A *write operation* must acquire an exclusive token
- while, a *read operation* can proceed with a shared or an exclusive token

The tokens act as a mechanism to ensure mutual consistency by creating conflicts among writes, and between reads and writes.
In the event of a failure, only the partition containing a token is allowed to access a copy of d.

ROWA vs QC

ROWA constraints

1. Inflexible favoring of read availability: the ROWA family of protocols implicity favors *read operations* by allowing them to proceed with only one copy, while requiring *write operations* to be carried out at all the up sites.
2. Inability to tolerate communication failures: the latter condition also mean that ROWA algorithms cannot permit write operations to succeed when it is not possible to communicate with an up site because of a network failure

Quorum Consensus or Voting

Quorum

QC methods (or voting methods) allow *writes* to be recorded only at a subset (a write quorum) of the up sites, so long as *reads* are made to query a subset (a read quorum) that is guaranteed to overlap the write quorum.

- This quorum intersection requirement ensures that *every read operation will be able to return the most recently written value.*

Voting

A site that partecipates successfully in an operation is said to have *voted* for it (hence the alternative name method).

Biggest advantage (great advantage!)

QC techniques mask failures, with no need for intervention in order to resume operation after network partitions are repaired and merged.

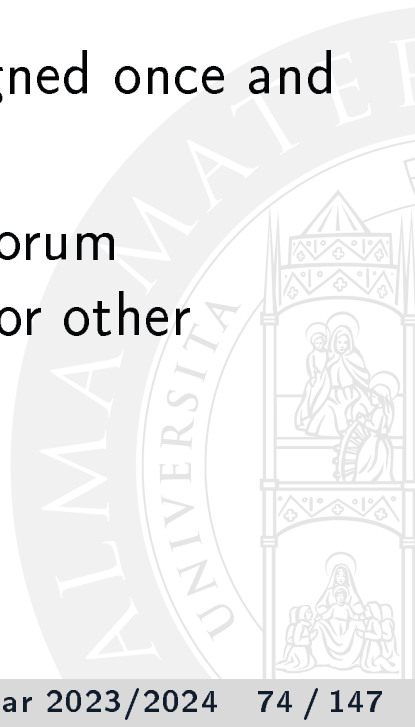
Static vs Dynamic

Definition

A read (or write) quorum is the minimal set of copies whose permissions are required for a read (or write) operation to proceed.

Quorums can be:

- static: as when they are specified by votes that are assigned once and for all at system startup time
- dynamic: if the sites are capable of reconfiguring the quorum specification (e.g. in response of failures, load changes, or other events)



Uniform Majority QC protocol I

Basic idea: a majority requirement

- The DDBS is modeled as a group of sites that *vote* on the acceptability of requests (e.g. query/update).
- An operation (i.e. *read or write*) succeeds if and only if a majority of the sites approve its execution.

But...

Not all the sites that vote for an operation need to carry it out their local copies; a *read operation* needs to be executed at *only one current copy*, while a *write operation* must be executed at a *majority of the copies*.

Uniform Majority QC protocol II

Pro

- A majority requirement ensures that there be an intersection between the read and write operations of two transactions and that the underlying *concurrency control protocol* detects the conflicts and permit *no conflict updates*
- Resiliency to both site and network failures is achieved

Cons

- The resiliency to failures, implies high read and update costs: at least, half of the n sites must participate, via voting, in every access of data.
- The method works in the presumption that a network failure partitions the sites into two groups (a majority partition and non-majority partition); repeated failures may splinter the system into many groups of sites, with none of the groups forming a majority.

Weighted Majority QC protocol I

The Weighted Majority QC algorithm generalizes the notion of Uniform Voting.

Algorithm

- Instead of assigning a single vote per site, each copy d_s of a data item d is assigned a *non-negative weight* (a certain number votes) whose sum over all copies is u.
- A data item d itself is assigned a read threshold (r), and a write threshold (w), such that:
  - r+w > u, and
  - w > u/2, only if version numbers are used to determine the most current copy.
    An alternative method that uses timestamps can have
    w <= u/2

Weighted Majority QC protocol II

Read or write quorum

A read (or write quorum) of d is any set of copies with a weight equal to at least r (or w).

- This additional constraint, provides greater flexibility in vote assignment, besides ensuring mutual consistency of the copies just as in majority consensus.

A versioning mechanism determines the currency of the copies:

- each copy is *tagged* with a version number
- initially set to zero.

Thus, any read quorums guaranteed to have a current copy

Weighted Majority QC protocol III

Concurrency Control Algorithm

if:

- a transaction T' reads a copy of d

then:

- it can be safely concluded that it reads it from a transaction T that last wrote this copy of d.
  - T wrote into a write quorum of d
  - T' read from a read quorum of d
  - every read and write quorum have a non-empty intersection
- T' will read the value written by T and not by some earlier transaction, thanks by the version number.

Weighted Majority QC protocol IV

Requirements

- No complicated recovery protocol: transactions will continue to ignore stale copies until they are brought current

Pro

- The algorithm is flexible: by alterning r, w and the weight, the performance characteristics and the availability of the protocol can be altered.

Weighted Majority QC protocol V

Cons

- Workloads with a high proportion of reads will not perform well under weighted majority QC, because of the need to poll multiple sites before every read operation.
- low fault-tolerance to site failure

Solution exists

e.g. By setting the r, w thresholds, and the individual degenerates into a simple ROWA.

But...

Choosing weights and thresholds can be difficult task for a system administrator!!

Weighted Majority QC protocol VI

Random Weights [Kumar, 1991]

Randomized method that iteratively adjiust these parameters until they reach near optimal values.

- The steady state condition (maximum availability) is defined in terms of the iterations of the algorithm and is assumed to be achieved once there is *no change in the best solution over a certain number of iterations*.

Why hybrid

Expensive QC

QC algorithm is expensive if communication failures are infrequent!

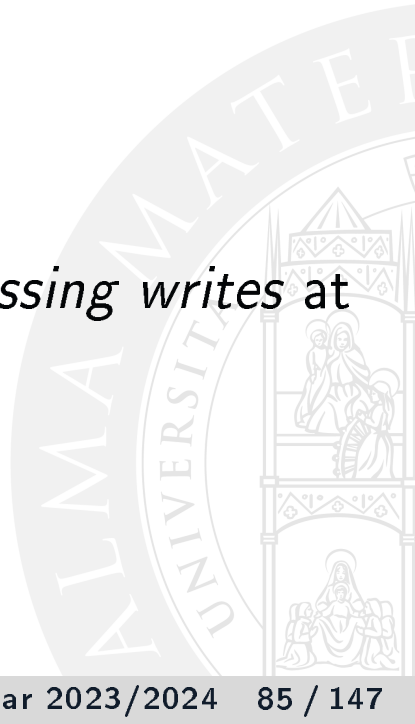
Hybrid ROWA/QC I

This hybrid ROWA/QC algorithm proposes a method to *reduce QC cost* which:

- adopts a ROWA strategy during reliable periods of operation and
- switches to the QC method during periods when there are site or communication failures.

A transaction:

- works under the ROWA method in its normal mode and
- switches to a failure mode when it becomes *aware of missing writes* at which point it obeys the QC protocol.



Hybrid ROWA/QC II

Detection of *missing writes* is important to:

- avoid performance degradation
- detect missing writes BEFORE access

MWL

A versioned missing write list (MWL) is associated with each copy of the data item.

- when a transaction T' access to a copy in a *conflict mode*, it becomes aware of the missing writes.
- T' propagates this MWL to all other sites that it visits

Upon recovery from site failure, a recovering site s has to eliminate the entry corresponding to d_s from the MWLs of all other copies.

Hybrid ROWA/QC III

Pro

The algorithm *maintains availability during failures*

Cons

But at the expense of *increased complexity and communication costs*.

Actually data replication algorithms I

Many of the more recent data replication algorithms emphasize: cost optimization, in addition to fault-tolerance.

One possible optimization

Savings in communication costs obtained by *decreasing the number of sites that need to be contacted* in order to ensure mutual consistency

- new protocols try to reduce the number of sites from n/2 (in the case of majority voting), to a lower number, by imposing a logical structure on the set of sites.

Actually data replication algorithms II

Matrix based algorithms

- $\sqrt{n}$ Algorithm
- The Grid Protocol
- Multidimensional Weighted Majority QC

Tree-based algorithms

- Tree Quorums
- Hierarchical Weighted Majority QC

The basis for most of these protocols is the generalization of quorums

As mechanism to achieve fault tolerance

Work in this area presents two fundamental approaches:

1. The first is modular redundancy, in which each component performs the same function
   - The replicas here are called active replicas
2. The second involves primary/standby schemes, where a primary component functions normally, periodically checkpointing its state to the secondary replicas, which remain on stand-by and take over in case the primary fails.
   - The replicas here are called passive replicas

RPC

RPC

The simplest mechanism for remote, and hence distributed, execution of programs is the remote procedure call.

- RPCs are capable of providing distribution transparency only as long as all systems are running correctly.
- help in the implementation of distributed threads of control between the modules
- provide *location transparency*

Failure of some of the sites causes a distributed program to crash at those sites, resulting in *partial failure* of the system, thereby violating transparency.

- Replication offers a method to mask these partial failures.

Modular Redundancy Replication I

Advantages

- simple and cost-effective alternative
- can be applied systematically to any application
- also, can be applied selectively to critical components

All these advantages are realizable *only if* user-transparency to the underlying process replication is achieved.

Modular Redundancy Replication II

Compared to the passive replica approach, the active replica approach:

Advantages

- eliminates complicated checkpointing procedures during normal operation
- a site failure need not induce any explicit failure recovery protocol
- better performance can be achieved by using a local replica (if available)
- leads to better utilization and load-balancing of system resources

All these advantages are realizable *only if* user-transparency to the underlying process replication is achieved.

Inconsistent execution

Definition

An inconsistent execution is a phenomenon where different replicated processes behave differently.

...problems

Inconsistent executions lead to two main problems:

- inconsistent process states
- inconsistent external behaviour (like non-deterministic program behaviour).

Consistency of Processes I

The simplest form of consistency between processes is no consistency at all. (e.g. *dynamic server squads in Yackos*).

But...

When replicated process are used as a means to achieve reliable program execution, some mechanism is required to maintain *mutual consistency* of the processes in order to achieve replication transparency.

Consistency of Processes II

The consistency requirements are based on two main models of replicated processes:

- In the Circus approach, processes are mutually consistent *if all the processes are deterministic* and if all of them, are in the same state.
  - Consistency is achieved using the *transaction concept*.
- In the Cloud approach, the problem of non-determinism is avoided by *committing only one* of the replicated processes and *copying* its state to other processes in order to maintained consistency

Circus approach I

- Programs are modeled as a collection of distributed module.
- Module packages the procedures and state information needed to implement a particular abstraction and separates the interface to that abstraction from its implementation.
  - Only one logical instance of a module
- Thread is the basic unit of execution in the modules:
  - A program begins as a *single thread* of control and,
  - *additional threads* may be created and destroyed using operation primitives.
  - A thread can execute in exactly one module at any given time but,
  - several threads can *concurrently* run in the same module

A thread moves from any given module →to the next, by making RPCs to, and returning from procedures, in the latter module.

Circus approach II

Definition: Deterministic module

Given a time interval I, during which a sequence of events have taken place, a module is said to be deterministic if the state of the program at the time that an event occurs, uniquely determine the event that follows I and the state of the module when that event occurs.

Circus approach III

Pro

- *Troupe consistency* is a necessary and sufficient condition for replication transparency.
  - A troupe is a replicated module, and each module is called a *troupe member*
- *Replicated procedure call*: is a generalization of RPC for many-to-many communication between troupes
  - RPCs are of the types: one-to-many, many-to-one, many-to-many
- Troupes remain consistent, if there is:
  - only a single thread of control,
  - in a globally deterministic replicated distributed system, and
  - if all troupes are consistent to begin with.

An inter-module procedure call results in a replicated procedure call from a client troupe to a server troupe.

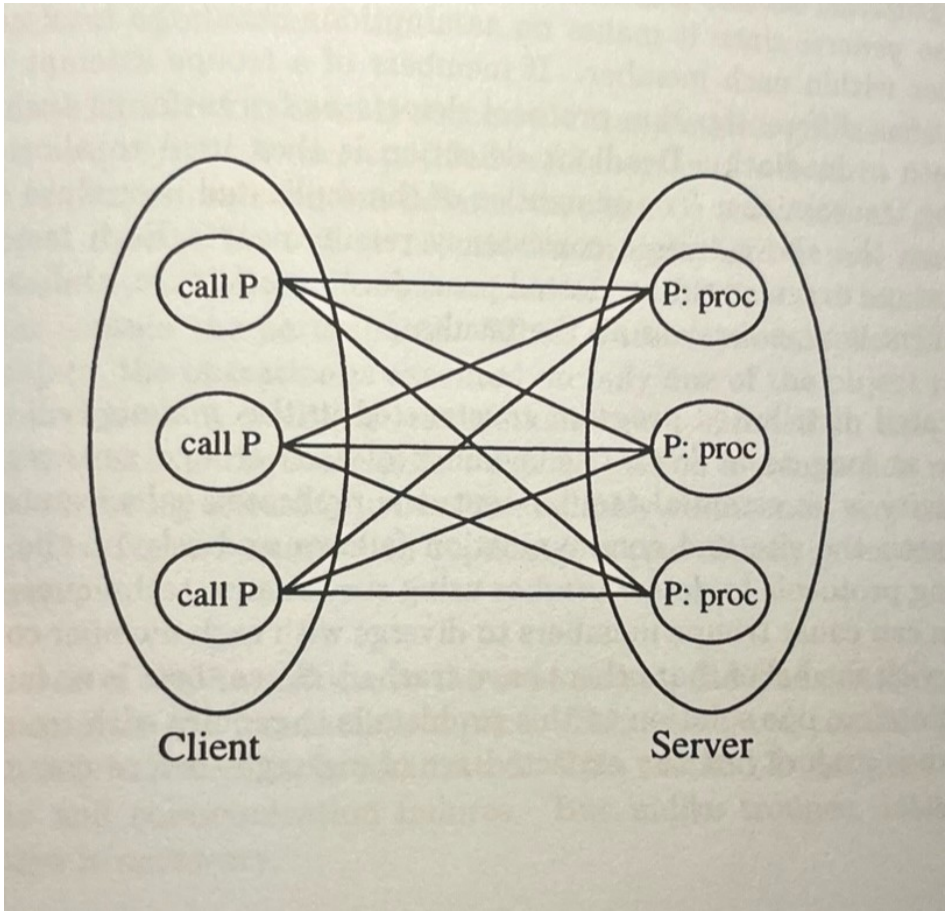


Figure: Replicated procedure call of type many-to-many in Circus

Cons of Circus approach

- The Circus approach requires the operation to be deterministic in order to maintain the members in a consistent state.

Clouds approach

Idea

The Clouds approach avoids non-determinism by committing *only one replica* of a replicated computation.

- The system is modeled as a collection of objects; each object replicated at several sites.
- Each obect replica is composed by:
  - shared state, which refes to the state of the object that captures the modifications made by successfully committed transactions;
  - code, read only object code;
  - buffered updates and locks, for transaction management within each object, to keep track of the updates made and lock acquired

Clouds approach

Workflow

1. A *root object* invokes a specific transaction
2. A transaction is executed as several concurrent transaction replicas
3. A user-specified *resiliency level* is used to determine the number of site/communication failures that can be tolerated.
   - implies that at least R transaction managers, and hence, sites must be participants in the protocol.
4. A *selection algorithm* is used to determine the sites that contain the participants
5. Access to the objects by multiple transactions is coordinated so that no two transactions may access the same object replica.
6. When a transaction manager is notified of its being selected, it can immediately commence processing the transaction; the fastest transaction replica gets committed, given that there are no failures.

Clouds approach

Pro

- The associated commit protocol ensures that only one replica is committed
- This method is designed to handle both site and communication failures

Cons

- Unlike troupes, inter-replica communication is necessary!

Passive replicas

1. Only one primary replica performs the computation, while
2. the rest monitor the state of the primary replica through some failure detection algorithm.
3. The primary has to periodically checkpoint its status to the secondary replicas so that,
4. one take over if and when the primary fails.

The Tandem $Non - Stop^{TM}$ Kernel

Is an early and most popular commercial implementation of the primary/standby mechanism.
The Tandem System is a multi-computer system with redundant hardware components.

As mechanism to achieve performance

Pro

- This approach of using *redundancy* to improve performance is particularly beneficial in systems with *abundant resources*.
- System with abundant resources are particularly common in *real-time environment*, and they are not to be expected in conventional systems.

In distributed object-oriented systems, components of the composite object may be distributed across multiple machine

so...Replication!

Replicating a composite object improves the availability and provides efficient reads

...but

Replication could be *expensive* in term of update and storage.

Selective Replication

Solution: Selective replication

Hence, replication should be selective.

- In distributed relational database systems (RDBMS) can be achieved by *fragmenting the relations* and *replicating* the horizontal and/or vertical fragments.
- Object-oriented database systems (OODBMS) provide selective replication using the *reference attributes* of the object
  - These reference attributes act as *pointers* to object replicas or to procedures

Replication of Objects

Requirements

1. Selection strategies: to choose the sub-objects to replicate
2. Replication schemes: to be used in the management of selected replicas

Four selective replication strategies

1. Replication of objects at particular levels in the object's hierarchical structure;
2. Replication of objects and sub-objects based on acess frequency;
3. Replication of objects capabilities;
4. Dynamic reconfiguration.

Three distributed composite object replication schemes

1. Class specific
2. Instance-specific
3. Instance-set-specific

Class specific

- A single replication scheme is specified for all objects of the same class
- Objects of a class are stored together and hence get replicated at the same sites.

When can be used

When a class is registered with the database system.

Characteristics

- (Con) Provides the least flexibility but, it also
- (Pro) has nominal cost in terms of replication management

Instance-specific

- The replication information is specified separately for each object (instance)

Characteristics

- (Pro) Provides maximum flexibility but,
- (Con) has higher cost in terms of replication management

Instance-set-specific

- Allows specifying different replication for different collections of objects belonging to the same class

Message Replication

The management of *data or process replication* → usually requires similar messages to be exchanged to all or some of the replicas.

Message basics

- *conform to standard communication protocols*
- *not guaranteed to be delivered within a maximum delay*
- a *common reference* could be useful

Message Replication forms

Simplest form

The simplest form of message replication is:

- to generate as many message replicas as required by the implementation of the data (process) replication protocols
- *by-product* replication
- no special message coordination scheme is needed

Other forms

The other two forms:

- Reliable Broadcast Protocol
- Quorum Multicast Protocol

do more than obey a higher level replication protocol!

Reliable Broadcast Protocol

Reliable Multicast or Broadcast Protocol

- *for reliable communication support*
- *simplifies replication in fault-tolerant distributed systems*
- *avoids several rounds of message exchange*
- only a single broadcast message (required for distributed operations)

The low-level message subsystem, or a higher level abstraction, provides for fault-tolerance and message delivery guarantees in the presence of communcation or replica failures.

This form contributes of the simplification of the management of replicated data/processes!

Quorum Multicast Protocol I

Quorum-oriented Multicast Protocol

- allows delivery to a *subset* of the destinations, selected according to availability, distance or expected data currency
- provides well-defined *failure semantics*
- can distinguish between **communication failure** and **replica failure** with high probability!
- pushes the most part of replication management, into an above-kernel (high-level communication)
- shifts even more responsabilities of replicated data management from the higher level protocols into the message subsystem

Quorum Multicast Protocol II

Advantages

- *Substantial reduction in the complexity* of data/process replication design and implementation
- Data replication provide *good performance* when quorum multicast are used

Quorum Multicast Protocol III

Ideal protocol

The ideal protocol would provide:

- *highest availability*, with
- *lowest latency and traffic*

Quorum Multicast Protocol IV

In practice

Trade-offs between these measures must be made:

- on a LAN, *broadcast messages* allow replication protocols to send requests to all replicas in one message, while
- in an internetwork, a separate message must generally be sent to each replica

So, the quorum multicast protocols:

- can *dynamically select* this subset to be closest available destinations
- *improve the performance* of subsequent communication

Impediments of replication

Three impediments to implementing replication schemas are:

1. Heterogeneity
2. Mobility
3. Scale

Heterogeneity

Replication in *heterogeneus environments* is difficult due to site autonomy, because:

- it's hard to enforce atomic actions in these environments
- *global consistency* needs to be enforced
  - is further complicated when data is replicated

Mobility

- Is another challenge that necessarily involves replications
- Ubiquitous access to data and information

New approaches

That support *weakly connected operations* by mobile users:

- on-demand replication
- on-demand caching
- cache replication

Scale

Classical replication

- assumes a "small" number of replicas, and
- assumes a "small" size system

Large-scale systems

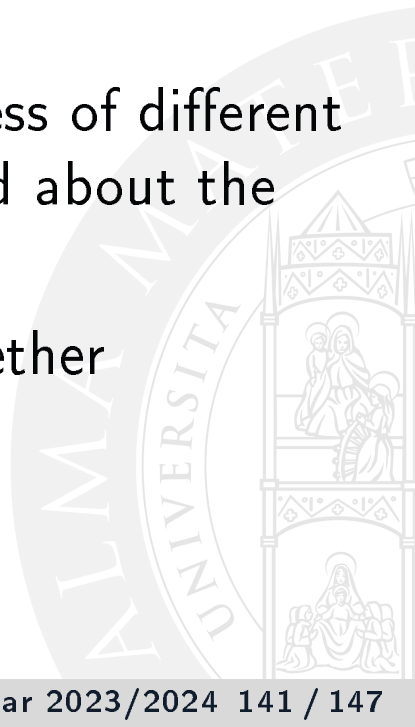
In *large-scale* systems classic replication does not work adequately!

- Due to performance degradation:
  - insistence on immediate consistency enforcement over a large number of replicas that must be accessed through a wide area network
- Due to assuming an inaccurate failure model:
  - in these large systems, the failure model is different (also because failures are almost always happening)
  - this mode of failure is called: *continuous partial operation*

Look at the future I

The field of **replication needs more experimental research!**

- further study of replication methods for computation and communication
- more work to enable replication in large-scale, real-time and heterogeneous environments
- resolve the conflict claims made about the relative success of different methods (in enhancing availability and performance) and about the validity of various assumptions
- best practices in how these tools may be integrated together

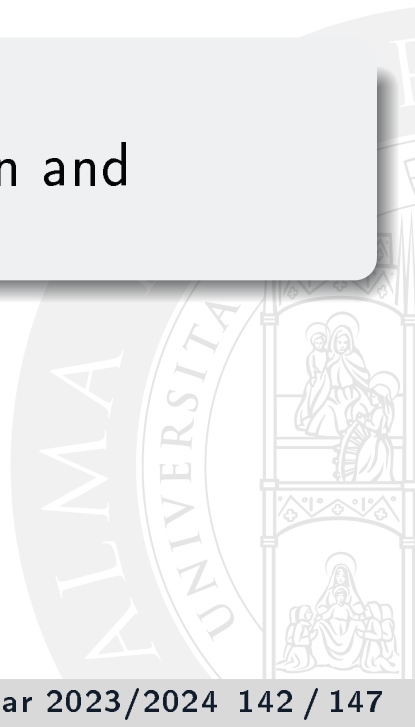


Look at the future II

It remains difficult to evaluate:

- the claims that are made by algorithm and system designers
- a consistent framework for comparing competing techniques

Only data replication has been studied intensely!
Relative little has been done in the replication of computation and communication.



Look at the future III

Example

One of the difficulties in evaluating replication techniques is the absence of a *commonly accepted failure incidence model*

- thus, the field needs empirical studies to monitor failure patterns
- with the purpose of constructing a simple model of typical failure loads

Failure load model I

A Failure Load Model

- should resolve such issues as that of *modeling the probability of sequences of failures*
- should include a variety of what might be called *failure incidence benchmarks*
- should include *availability metrics that are readily measurable*
- should clarify *how reconfiguration algorithms should exploit information about failures that have been detected and diagnosed*

Until this..

The most credible alternative is to construct testbed systems:

- for the specific purpose, of evaluating the relative *performance* achieved by different replication techniques
- with integration of a number of different algorithms to maintain consistency
  - while achieving *replication*, *reconfiguration*, *caching*, *garbage collection*, *concurrent operation*, *replica failure recovery and security*

Lots of work needs to be done...

Replication

Replication in Distributed Systems

Arianna Soriani
arianna.soriani@studio.unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna

Academic Year 2023/2024

References

[Charron-Bost et al., 2010] Charron-Bost, B., Pedone, F., Schiper, A., and editors (2010).
Replication. Theory and Practice, volume 5959 of Lecture Notes in Computer Science.
Springer, Berlin, Heidelberg
.

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

[Kumar, 1991] Kumar, A. (1991).
A randomized voting algorithm.
.

[Lamport, 1987] Lamport, L. (1987).
Dec src bulletin board.
.

[Oki and Liskov, 1988] Oki, B. and Liskov, B. (1988).
A general primary copy method to support highly available distributed systems.
.

[Raab, 1992] Raab, L. (1992).
Effects of replication on availability in distributed database systems.
.