Confiabilidade de Sistemas Distribuídos
Dependable Distributed Systems

DI-FCT-UNL, Nuno Preguiça

Lect. 2
State-machine replication

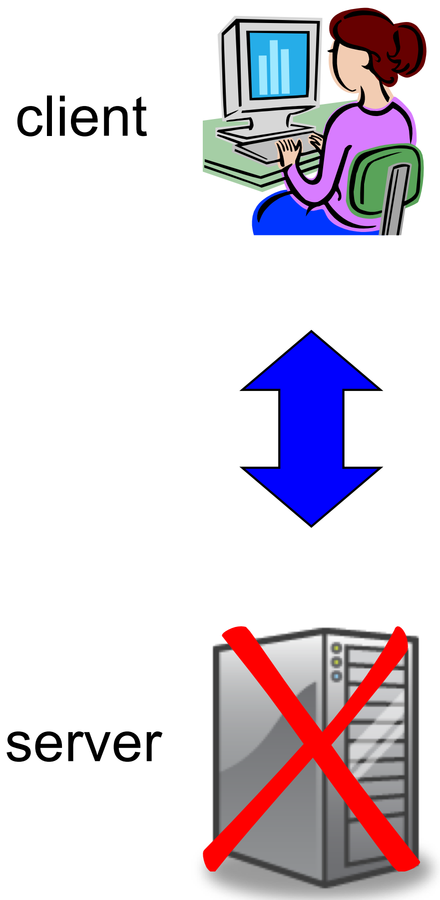
2018/2019, 2nd SEM

MIEI
Mestrado Integrado em Engenharia Informática

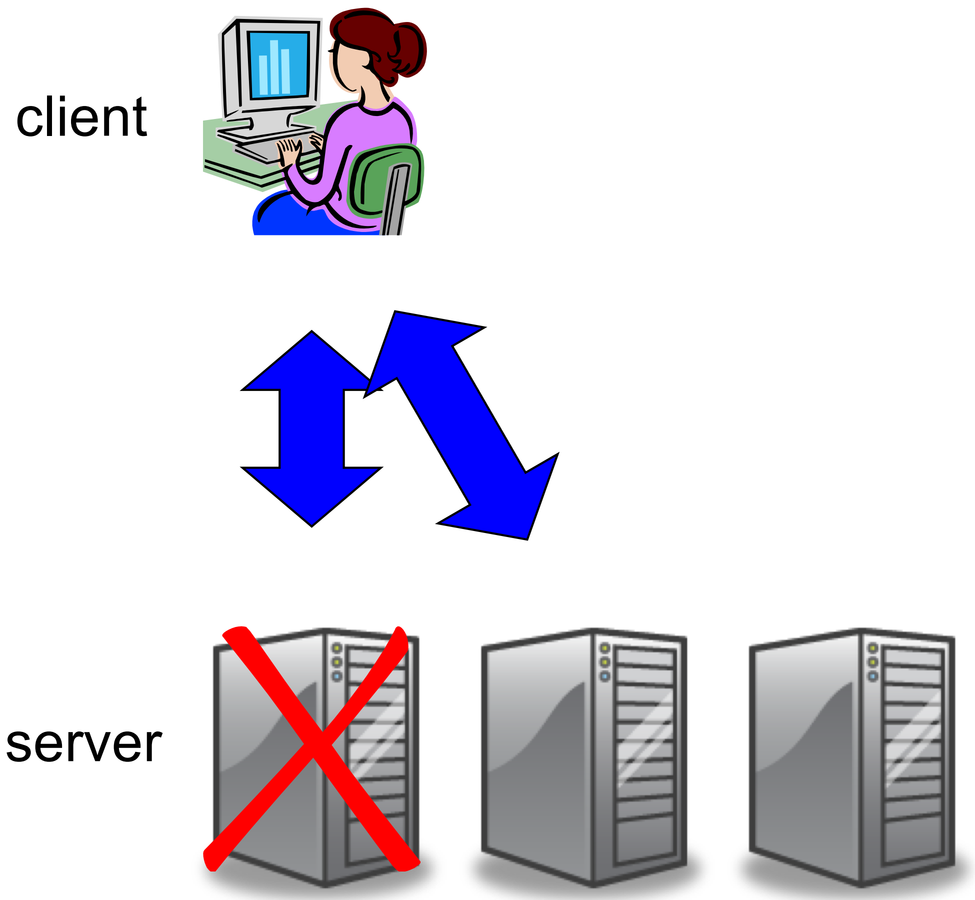
Outline

- Replication as basic mechanism for dependability
- Replication models
- Consensus
- Paxos

What to do in a crash fault?



What to do in a crash fault?



What to do in a crash fault?

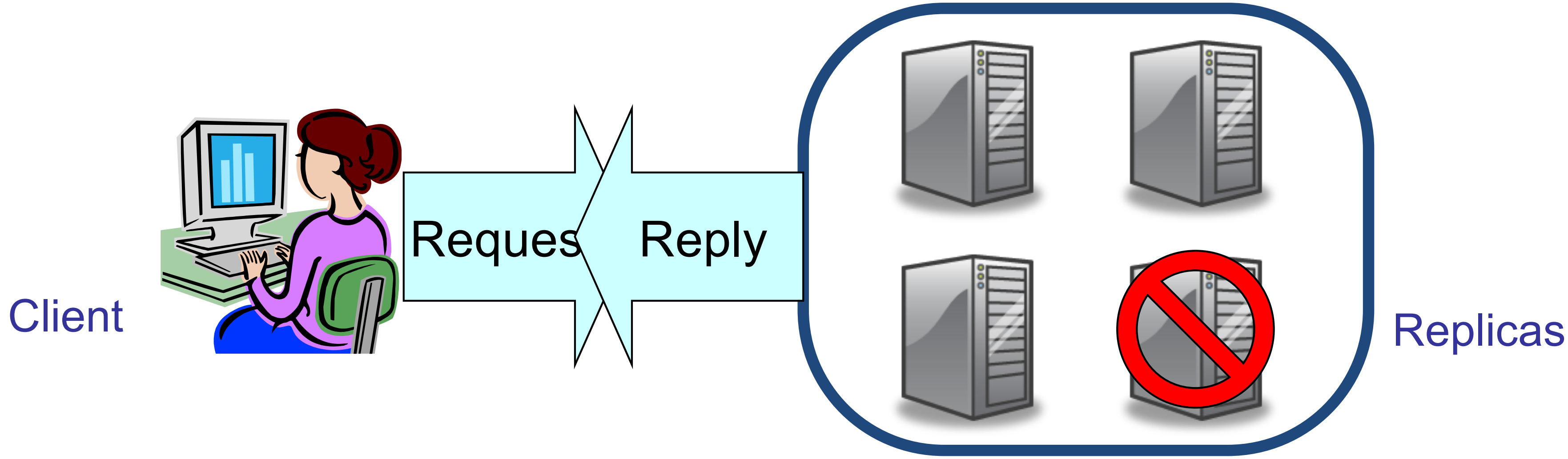
- If the service (and data) are replicated in multiple machines, it should be possible to tolerate faults

Replication models: read/write register

- Each server (replica):
  - Maintains a copy of the service state
  - Exports two operations:
    - read() – returns value previously written
    - write(val) – writes val, returning when operation is completed

Read/write register replication

1. Service is replicated
2. Operations execute in a quorum of replicas and provide the illusion of a single replica (atomicity)



Quorum system

- Given a set of replicas $P=\{p_1,p_2,\ldots,p_n\}$, a **quorum system** is a set $Q=\{q_1,q_2,\ldots,q_m\}$ of ubsets of P, such that $\forall i,j,\ q_i \cap q_j \neq \emptyset$

Majority

- All sets of the quorum system must include more tha half of the replicas
  - Given $n = | P |$, $\forall q_i$, $| q_i | > n / 2$

- Properties
  - All operations need to access the same number of replicas

Read-write quorum system

- A **read-write quorum system** is a pair of sets $\mathbf{R=\{r_1,r_2,\ldots,r_m\}}$, $\mathbf{W=\{w_1,w_2,\ldots,w_r\}}$, of subsets of P, such that :
  - $\forall i,j,\ r_i \cap w_j \neq \emptyset$ (read intersects write)
  - $\forall i,j,\ w_i \cap w_j \neq \emptyset$ (write intersects write)

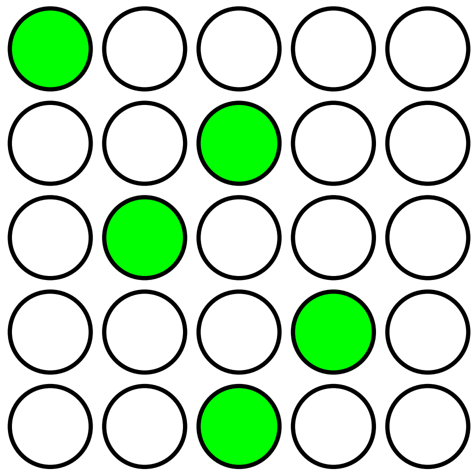
Read one / write all

- Every single replica is a read quorum, all replicas are included in the write quorum

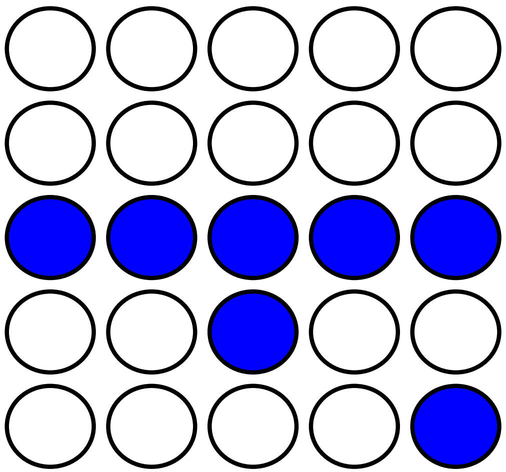
- Properties
  - Very light read; very heavy writes

Other quorum systems?

Grid



Read quorum:
One element in each line



Write quorum:
One complete line + one element in each line below

Grelha

- Properties:
  - Quoruns grow sub-linearly with the number of nodes: O(sqrt(n))
    - Load grows sublinearly with the number of request
  - Allows to balance the dimension (and availability) of read and write quoruns

Algorithm ABD [Attiya, Bar-Noy, Dolev]

- Assumptions: asynchronous system, reliable channels
- Requires 2f+1 replicas to tolerate f crash faults
  - Safety always guaranteed
  - Liveness only in execution with less than f faults

ABD: State and write algorithm

- State
  - val_i → value of the variable, initially v0
  - tag_i → pair <number of sequence,id> initially <0,0>
    - <s1,i1> > <s2,i2> iff s1 > s2 || (s1 == s2 & i1> i2)

- Client c : Write(v)
  - Step 1:

  Send(<read-tag>) to all processes (or to a quroum)

  Wait for a quorum Q of replies

  Let seqmax = max{sn : <sn,id> $\in$ Q }
  - Step 2:

  Send(<write(<seqmax+1,c>,v)>) to all processes (or to a quroum)

  Wait for a quorum of acks

ABD: Algorithm for replica i

- on_recv(<read_tag>)
  - Return <tag_i>

- on_recv(<write(new-tag,new-val)>)
  - If new-tag > tag_i then
    - tag_i = new tag
    - val = new-val
  - Return ack

- on_recv(<read>)
  - Return <tag_i,val_i>

ABD: Algorithm for read

- Client c : Read()
  - Step 1:

  Send(<read>) to all processes (or to a quroum)

  Wait for a quorum Q of replies

  Let <tagmax,valmax> $\in$ Q be the reply with largest tagmax

  - Step 2:

  Send(<write(tagmax,valmax)>) to all processes (or to a quroum)

  Wait for a quorum of acks

  Return valmax

Is all this complexity necessary?

- How does ABD protocol addresses the following challenges?
  - On concurrent writes, it is necessary to decide which value to keep
  - After a read returns some value, a read executed after must not return an older value
    - Note that reads execute concurrently with writes that are being executed and may fail in the middle of execution

Replication models: state-machine replication (SMR)

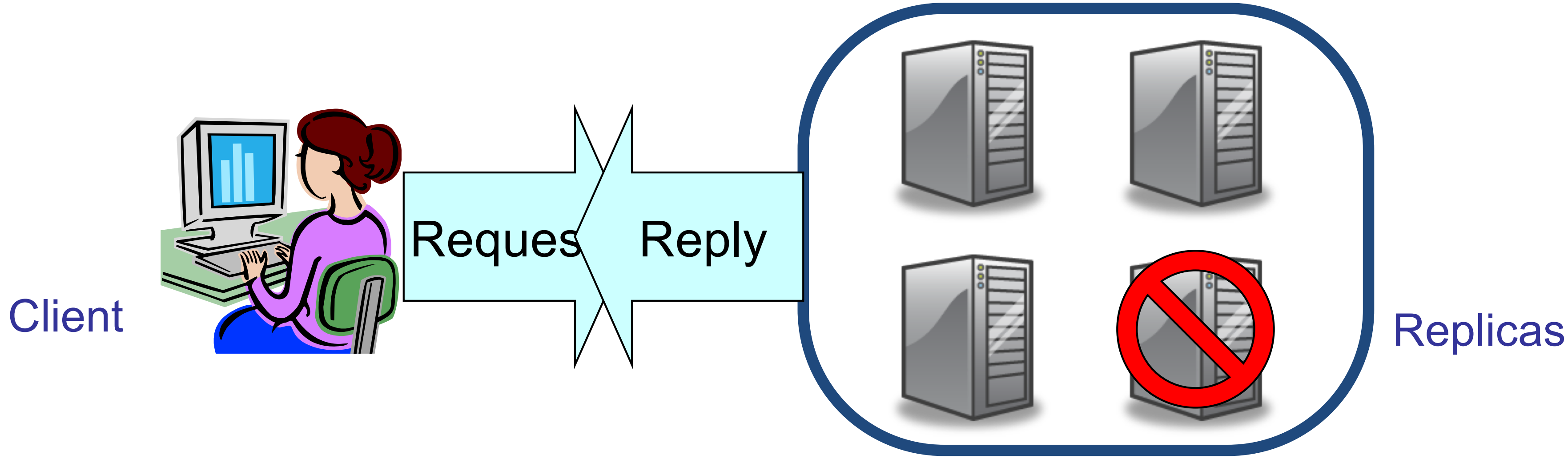
- Each server (replica):
  - Maintains a copy of the service state
  - Exports a set of operations O
- Each operation:
  - Has arguments (input)
  - Generates a result (output)
  - Makes a state transition in the server (i.e. change its internal state)

Determinism

- An operation is deterministic if the result and state transition it generate depends exclusively of the initial state and the operation arguments.

State machine replication (SMR)

1. Service is deterministic (i.e., all operation are deterministic)
2. Service is replicated
3. All correct replicas execute the same sequence of operations



Central requirement for SMR

All correct replicas execute the same sequence of operations

- Necessary to decide the order of execution of operations

Consensus

- Inputs: each process has its initial proposal in variable v_i
- Outputs: each process has an output variable $decision_i$, initially *null*
- C1 [Validity] If all processes have $v_i = v$, then v is the only allowed output
- C2 [Agreement] Two correct processes cannot decide different values
- C3 [Termination] All correct processes eventually decide
- C4[integrity] If a correct process decides v, then v was the initial proposal of some process

Central requirement for SMR

All correct replicas execute the same sequence of operations

- Necessary to decide the order of execution of operations
- Protocol:
  - Servers run a consensus protocols to decide the next operation to execute

FLP result

- There is no deterministic protocol to solve consensus in an asynchronous system in which a single process can fail by crash
  - Fisher, Lynch, and Paterson. Impossibility of distributed consensus with one faulty process. JACM, Vol. 32, no. 2, April 1985, pp. 374-382

Does this mean that SMR is a good idea that cannot be implemented in practice?