Three-tier Software Replication

Roberto Baldoni

University of Rome “La Sapienza”

Dipartimento di Informatica e Sistemistica

baldoni@dis.uniroma1.it

43rd Meeting of IFIP Working Group 10.4

Sal - Cape Verde 4-7 Jan. 2003

The Problem

"How to Increase the availability of a service ensuring strong replica consistency when clients and server replicas are **deployed** over the big Internet?"

Technique:

Software replication

System Model:

Crash failures, reliable channels

Motivation | 3T architecture | IRL

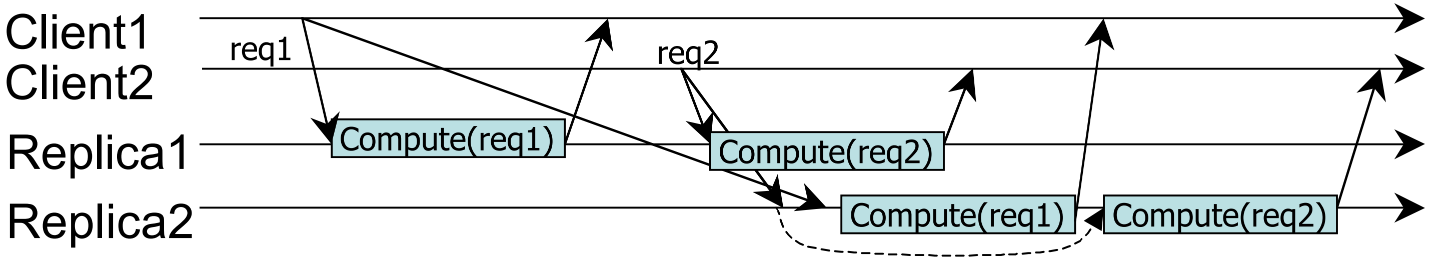
The technique: Software Replication

To maintain strong replica consistency (linearizability), it suffices:

- Atomic updates
- Ordering

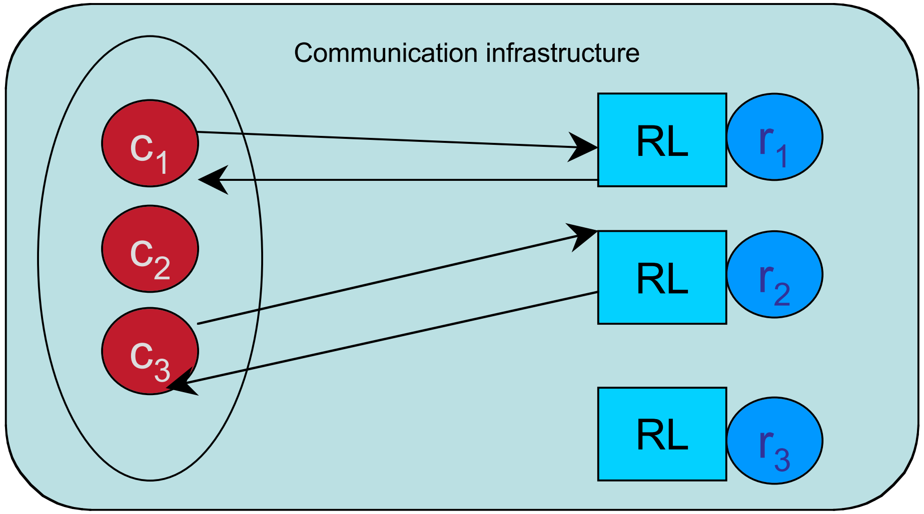
For example, Active replication

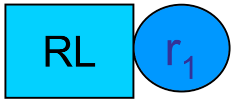
- Replicas must be deterministic
- All replicas process the same sequence of requests before failing (**agreement problem**)



2Tier (2T) Replication

- The replication logic (RL) is tightly coupled with replicas
- A wide set of instruments (*group toolkits*) is available to implement RL



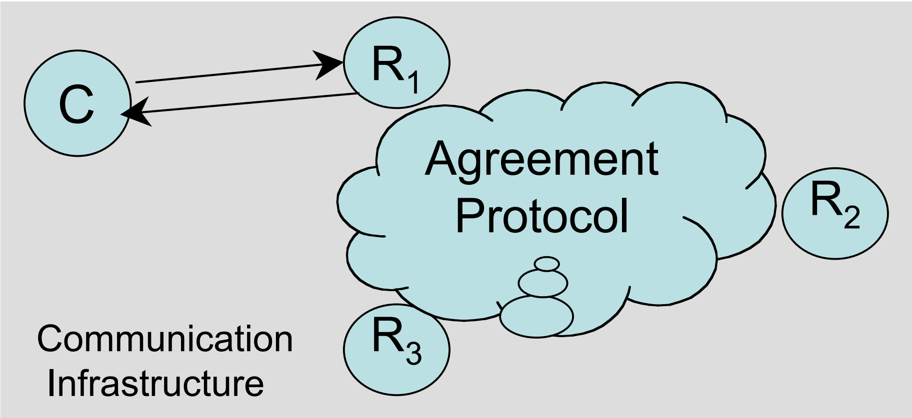


The replica is in charge of
•Request ordering (algorithms implemented by RL)

•Ordered Request execution

Big Problem in WAN (the big Internet): instability of the underlying system

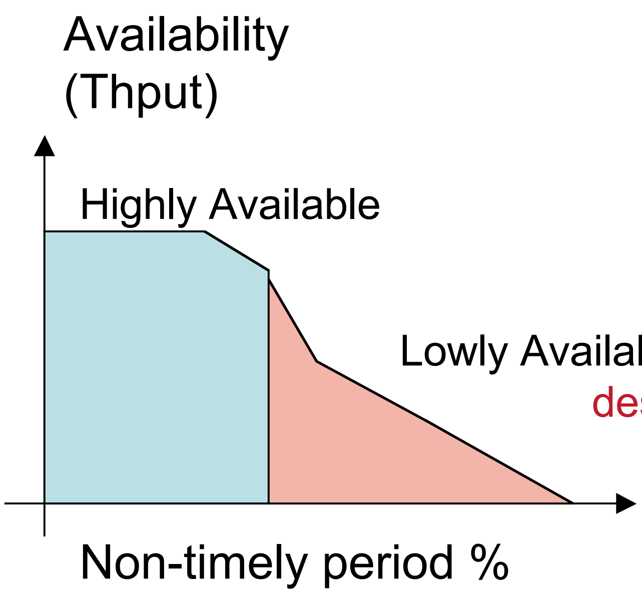
Implementing 2T Replication



A single "slow" processor or channel can slow down the whole protocol delaying the reply to the client

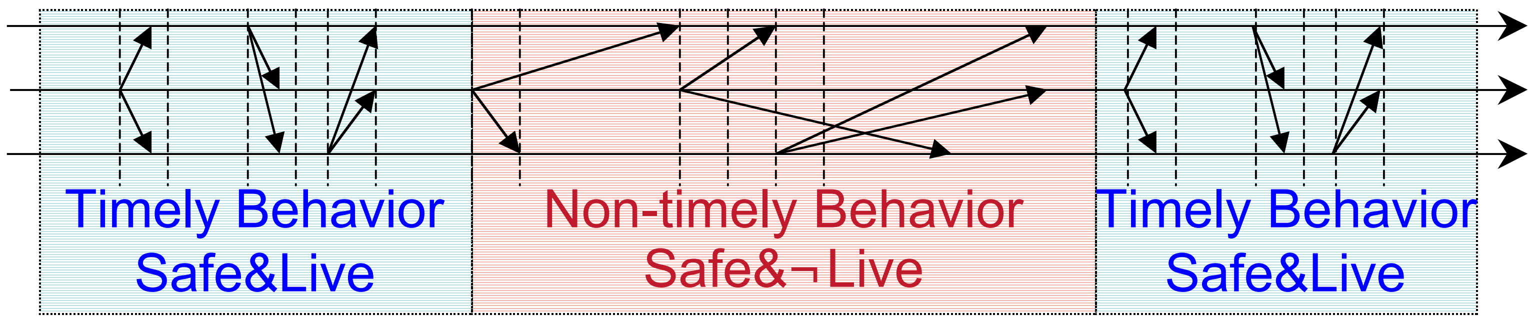
This injects asynchrony in the system causing instability (i.e., non-timely) periods

Birman (SPE99) points out problems of group communication in presence of asynchrony sources



Agreement requires Partial Synchrony

- The system alternates between timely and non-timely periods



Solving agreement problems require algorithms:

1. Always guarantee safety
2. Liveness is guaranteed during timely periods:

Looking at the big Internet

- Internet is highly unavailable (99,9% or 99,99%)
  - Prevent selected pairs of host to communicate about 1,2% of the time [Dahlin-Bharat-Gao-Nayate Trans. on Netw.]
  - Agreement protocols run very slowly [Bakr-Keidar PODC2002]
- This is perceived at end-to-end as unpredictable message transfer delays
- This causes non-timely periods
- In a 2T replication scheme, replicas continuosly loop in getting agreement

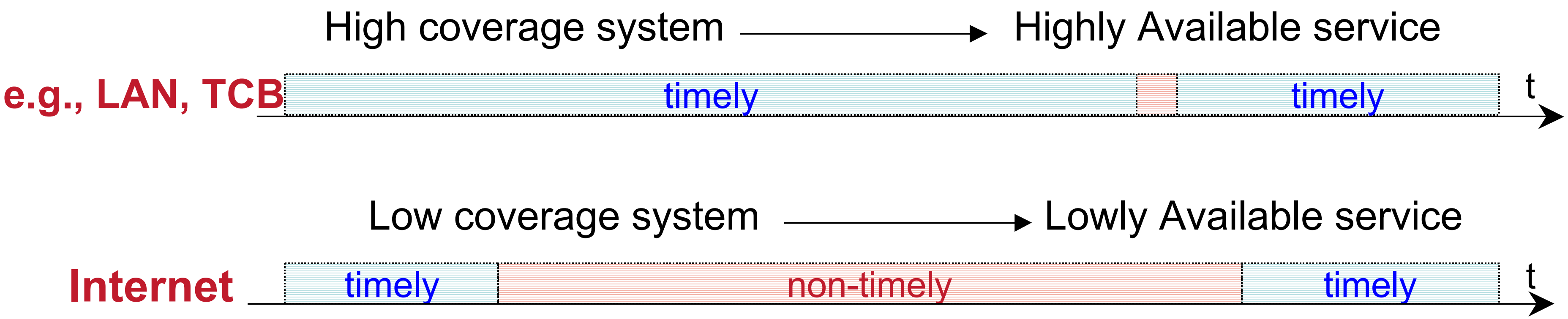
Looking at the big Internet

existing solutions to software replication face instability

- by weakening consistency guarantees, e.g.:
  - Eventual consistency
    - Partitionable groups, IceCube, Bayou, Lazy Replication, DNS etc.
    - Client can receive incorrect results wrt linearizability
  - Probabilistic guarantees (e.g. Bimodal multicast, epidemic diffusion etc.)
- by proposing efficent implementations aimed at improving the resiliency to instability periods
  - E.g. Moshe, spread

Looking at the Big Internet

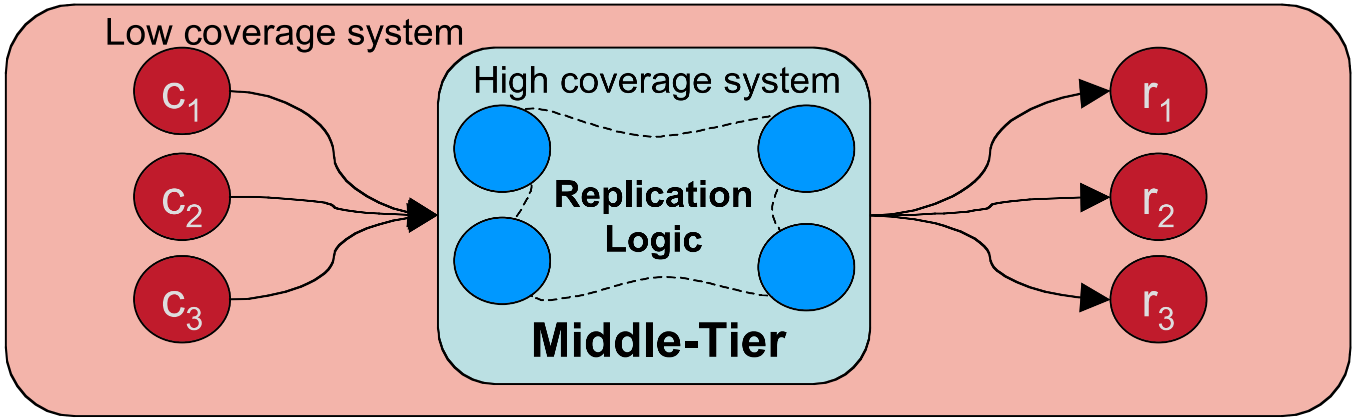
- service's availability *mainly* depends on the percentage of timely periods experienced by the system (i.e., on the coverage of partial synchrony assumption)



•Run critical tasks (e.g. Agreement) on a system with high coverage of partial synchrony assumption (e.g. a LAN)

3Tier Replication

"How to Increase the availability of a service ensuring strong replica consistency when clients and server replicas are deployed over the big Internet?"

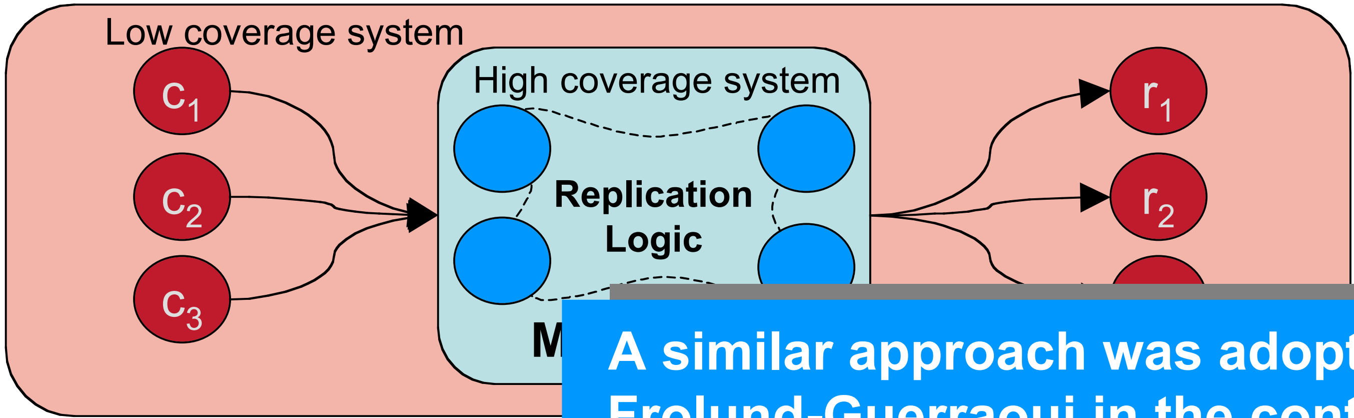


- **Run agreement protocols efficiently in a middletier under high coverage guarantees (e.g., a LAN)**

- **The middle-tier propagates clients' requests attaching the information necessary to each replica to ensure strong replica consistency without extra coordination**

3Tier Replication

"How to Increase the availability of a service ensuring strong replica consistency when clients and server replicas are deployed over the big Internet?"



A similar approach was adopted by Frolund-Guerraoui in the context of transactional systems

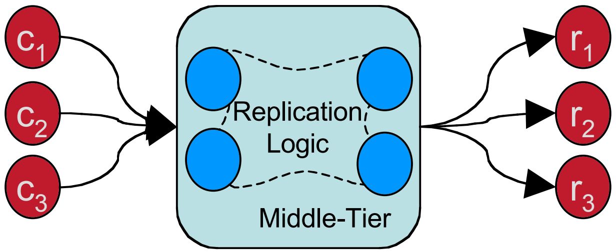
Middletier entities have to:

1. **Assign a sequence number to each replica**
2. **reliable delivery of each request to replicas**

1. **Each Replica has to:**
   1. **Ensuring the ordered request execution (similar to a reliable FIFO channel)**
   2. **Send the result to the client**

3T Software Replication

Active/passive/semipassive vs determinism/non-determinism of replicas

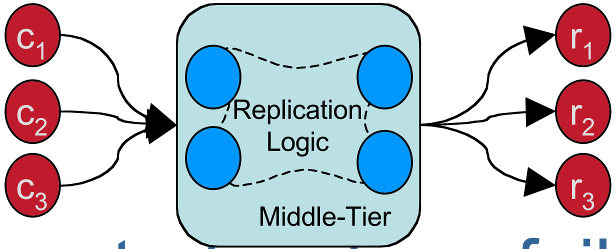


assume to tolerate *f* replicas failures, If replicas are deterministic, the middle-tier has to:

1. Take a #seq for the request
2. Forward the request to f+1 replicas
3. waits for the first result before forwarding the result to the client
4. Update the remaining n-(f+1) replicas

3T Software Replication

Active/passive/semipassive vs determinism/non-determinism of replicas

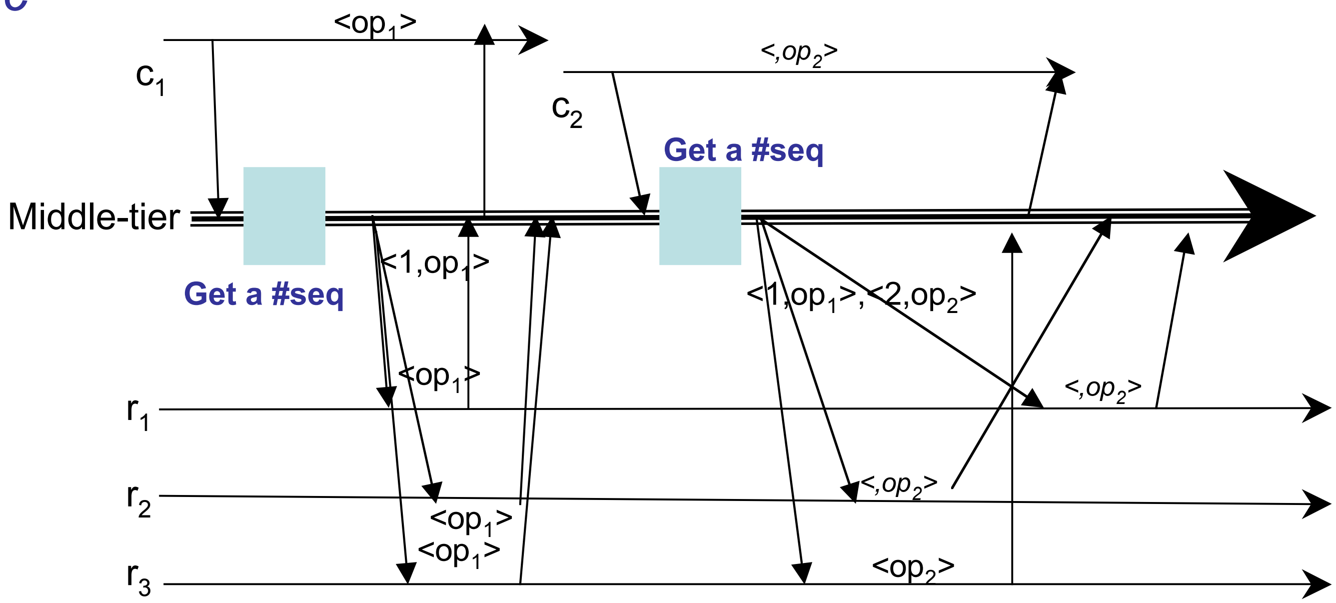


assume to tolerate *f* replicas failures, If replicas are non-deterministic, the middle-tier has to:

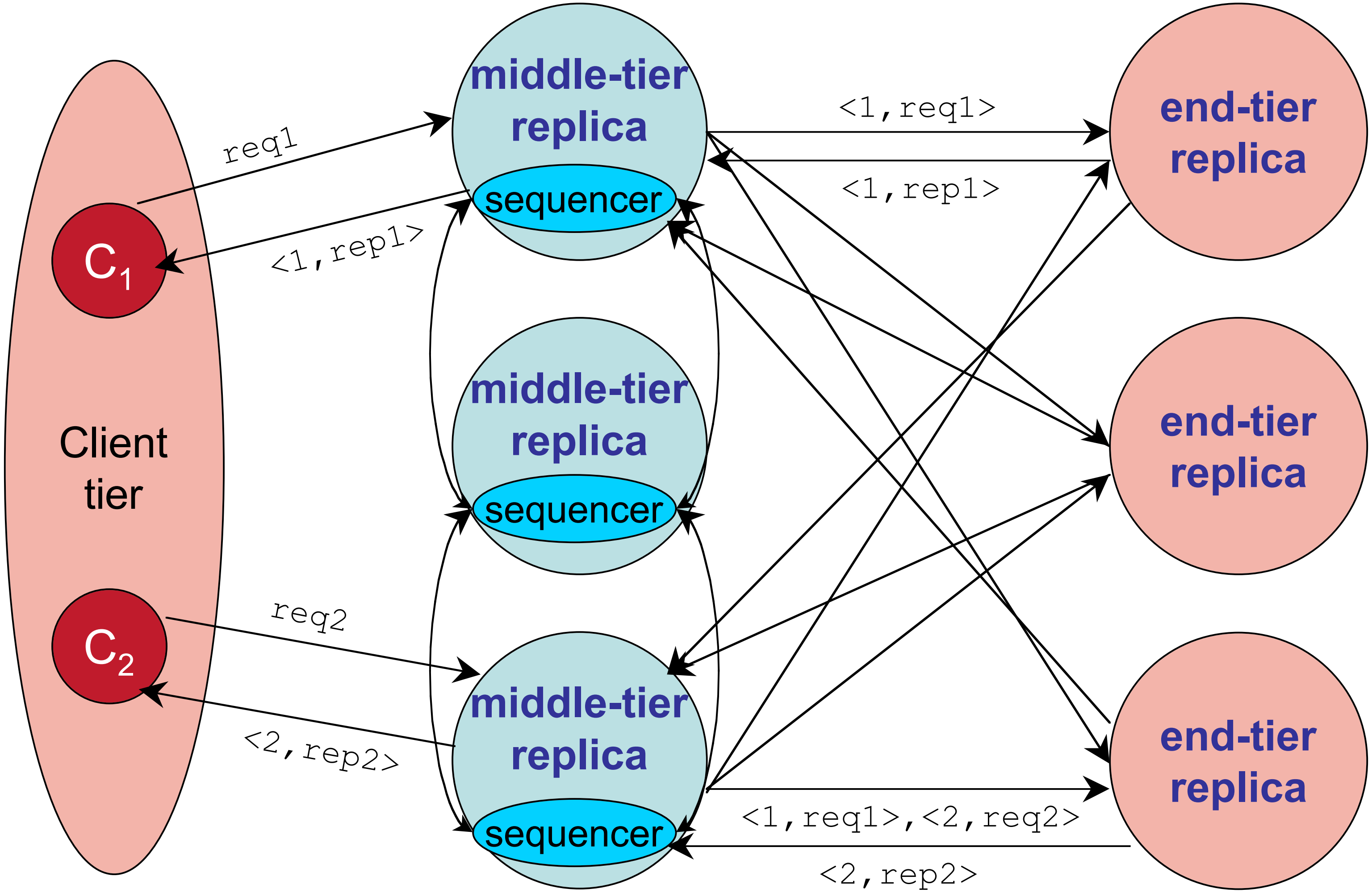
1. Take a #seq for the request
2. Forward the request to f+1 replicas
3. waits for the first result before forwarding the result to the client
4. Update the remaining n-1 replicas

3T implemented architecture (IRL)

- Middle-tier
  - –All middle-tier replicas can accept client requests <u>concurrently</u> to maximize availability
  - –Request ordering is based on a <u>distributed fault-tolerant sequencer service</u>
- *Server Replicas*
  - –*Deterministic*
  - –*f*= n-1



Protocol Overview

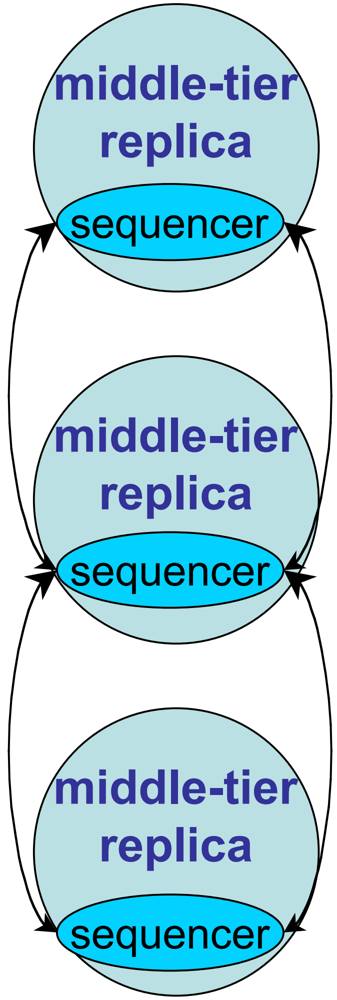


Sequencer overview

- Middle-tier replicas guarantee reliable delivery of client requests ordered by the *sequencer*.

- Sequencer service guarantees
  - each client request is assigned to **at most** one sequence number
  - **no** two client requests have **the same** sequence number
  - **sequence numbers are consecutive** (no “holes”)

- The sequencer encapsulates the agreement problem isolating the need of high coverage



Sequencer overview

Choosing the right total order multicast primitive

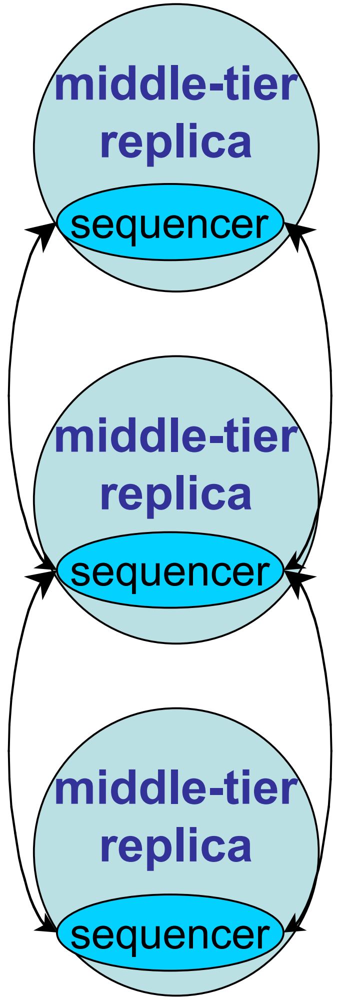
...... very difficult task

Property required by the sequencer:

Uniform total order with prefix order informally

“Each non-correct process delivers the same sequence of messages of a correct process till it crashes”

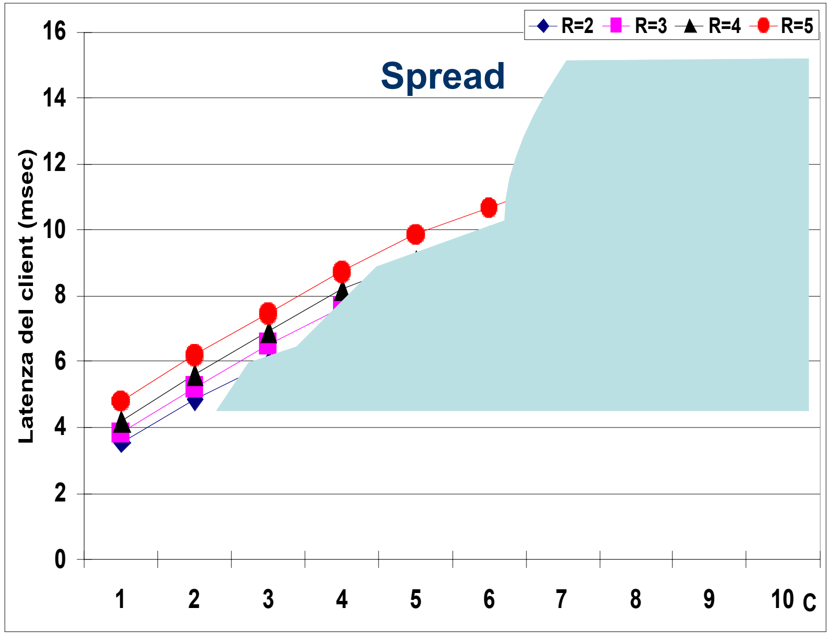
Implemented by Spread and Javagroups



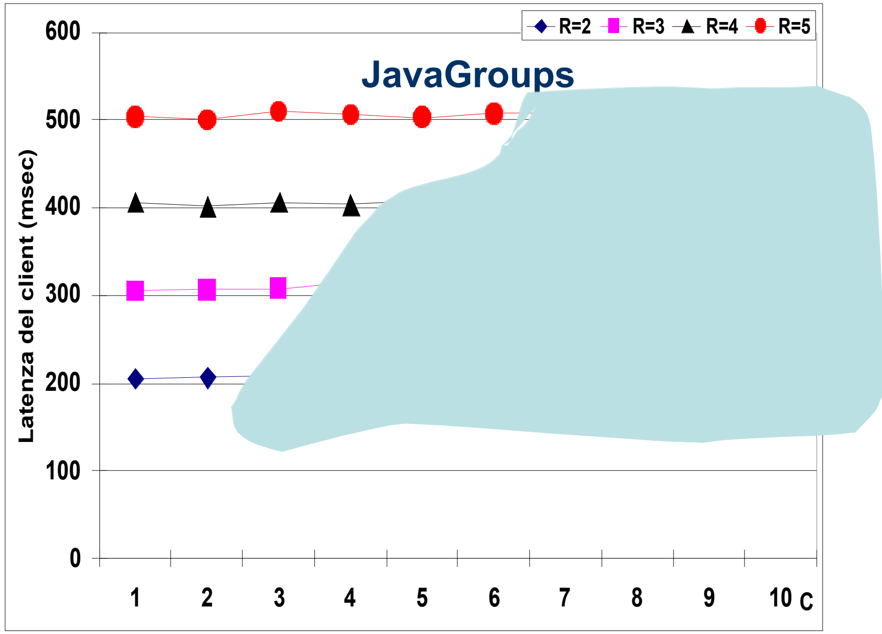
nSequencer Performance study

Extracted from a 2T replication performance study

Client Latency

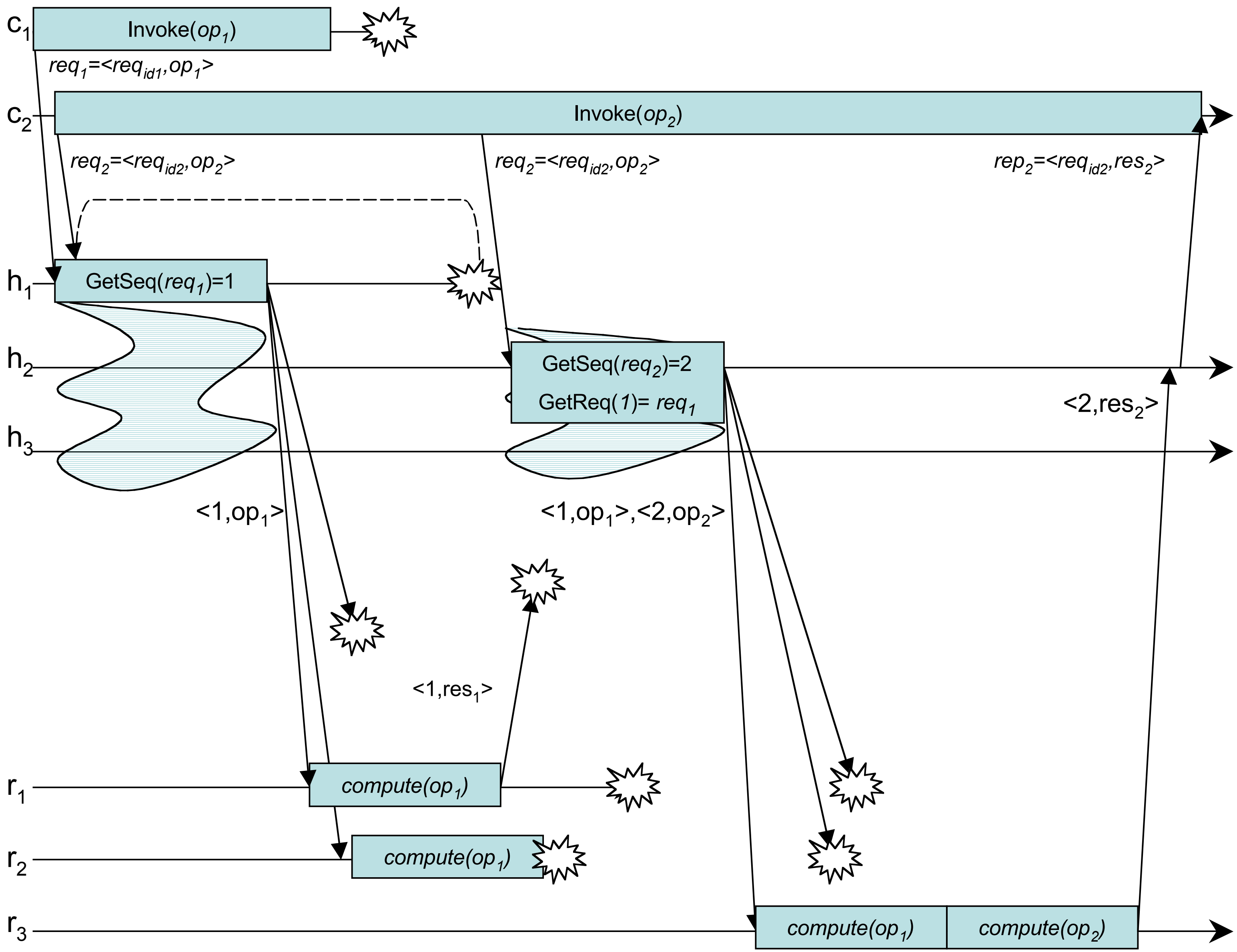


Client Latency

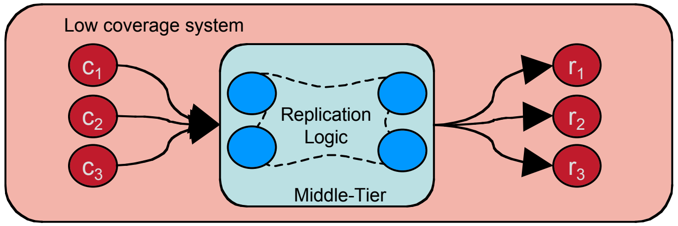


- #c <= #R
- IRL implementation is based on spread group toolkit

Run with failures

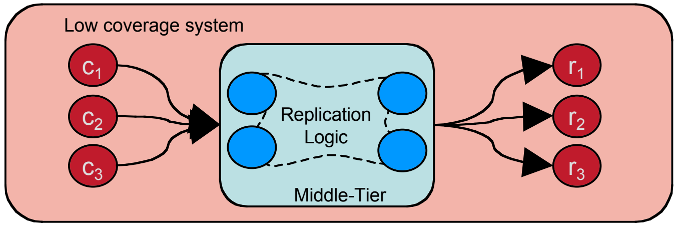


On the failure detection



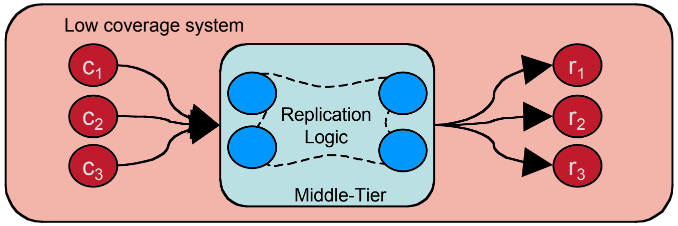
- Clients implements a simple retransmission protocol
- Middletier embeds the group toolkit failure detection system
- middletier implements a simple retransmission protocol

Optimizations



- Packing requests sent to an end-tier replicas into a message
- Reducing the message size by exploiting the end-tier replicas reply
- Client invocation semantic
- Bounding the size of the memory used by the sequencer

3Tier Replication



Advantages

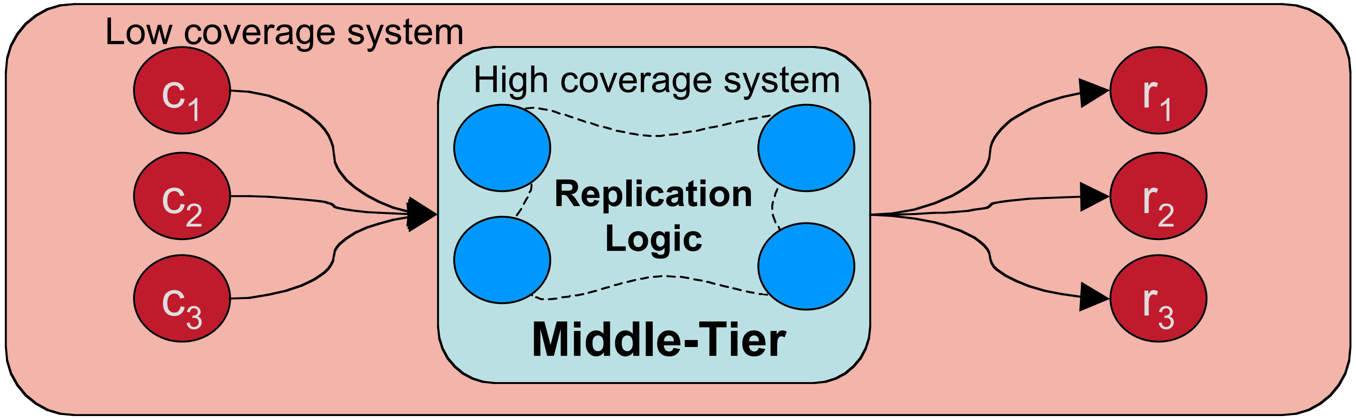
- Loose client and replica coupling
  - Clients and replicas are loosely coupled i.e., they do not interact among them
- Decoupling of service availability from data survivability
  - Data stored in replicas remain available despite middle-tier overall failures (which prevent service availability)
- Middle-tier extensibility
  - The introduction of further functionality e.g., load balancing, and tolerating malicious fault models, is simplified

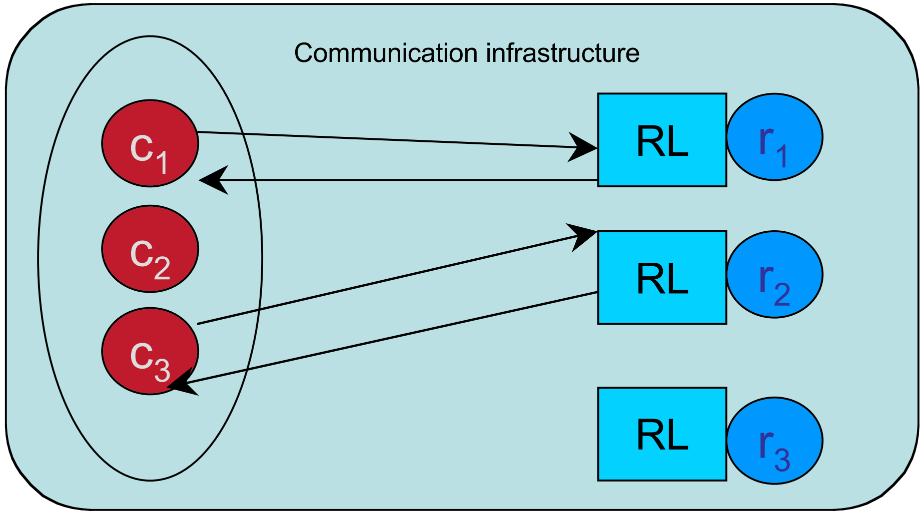
Drawbacks

- An additional hop in each client/server interaction
- *Service availability depends on the availability of the middle-tier*

On Service availability

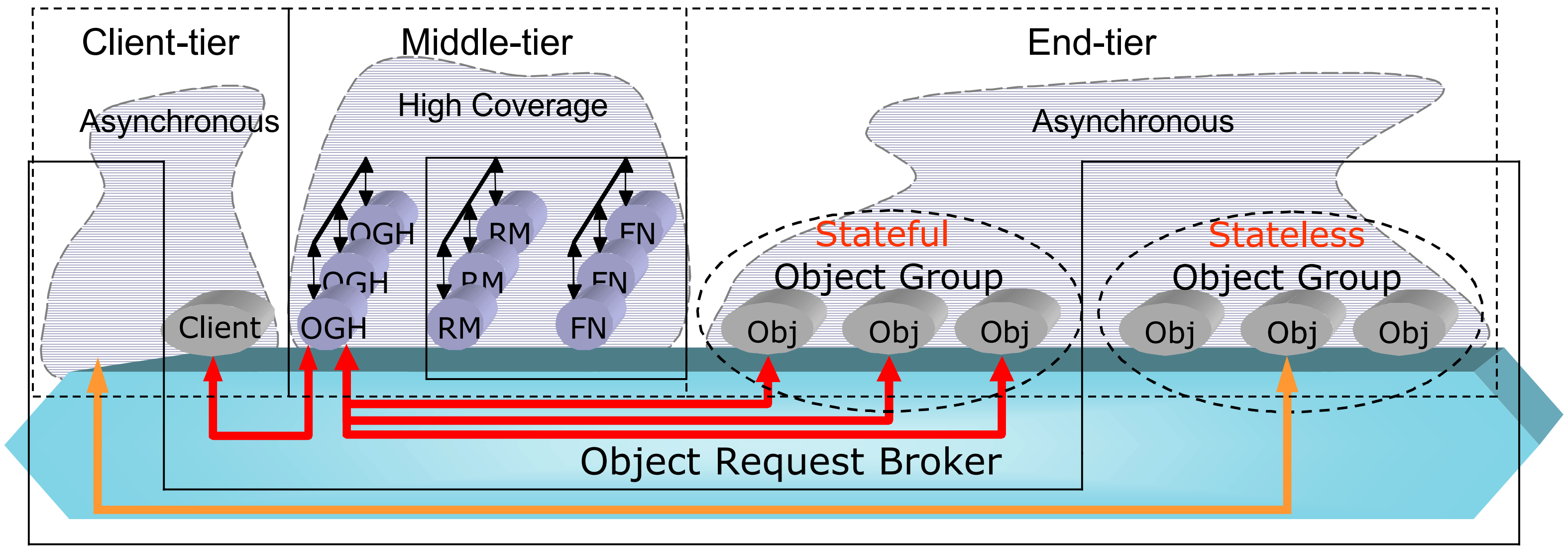
"centralized" vs "distributed"



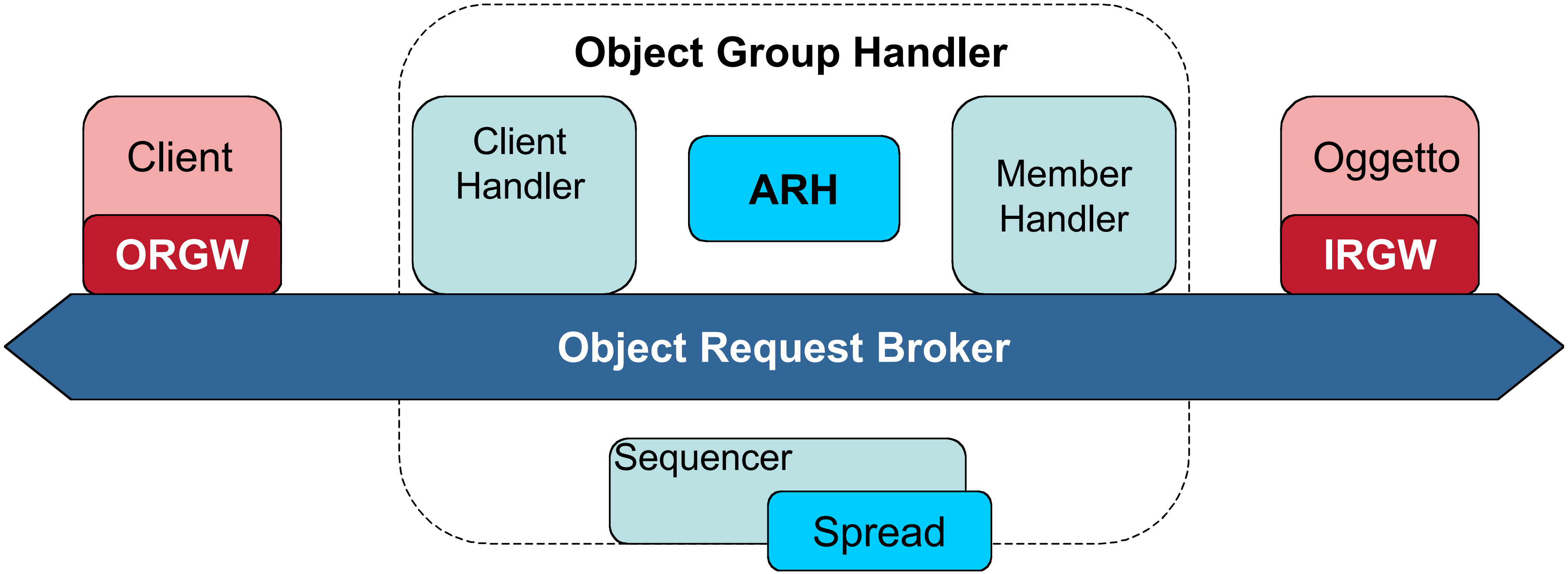


Interoperable Replication Logic

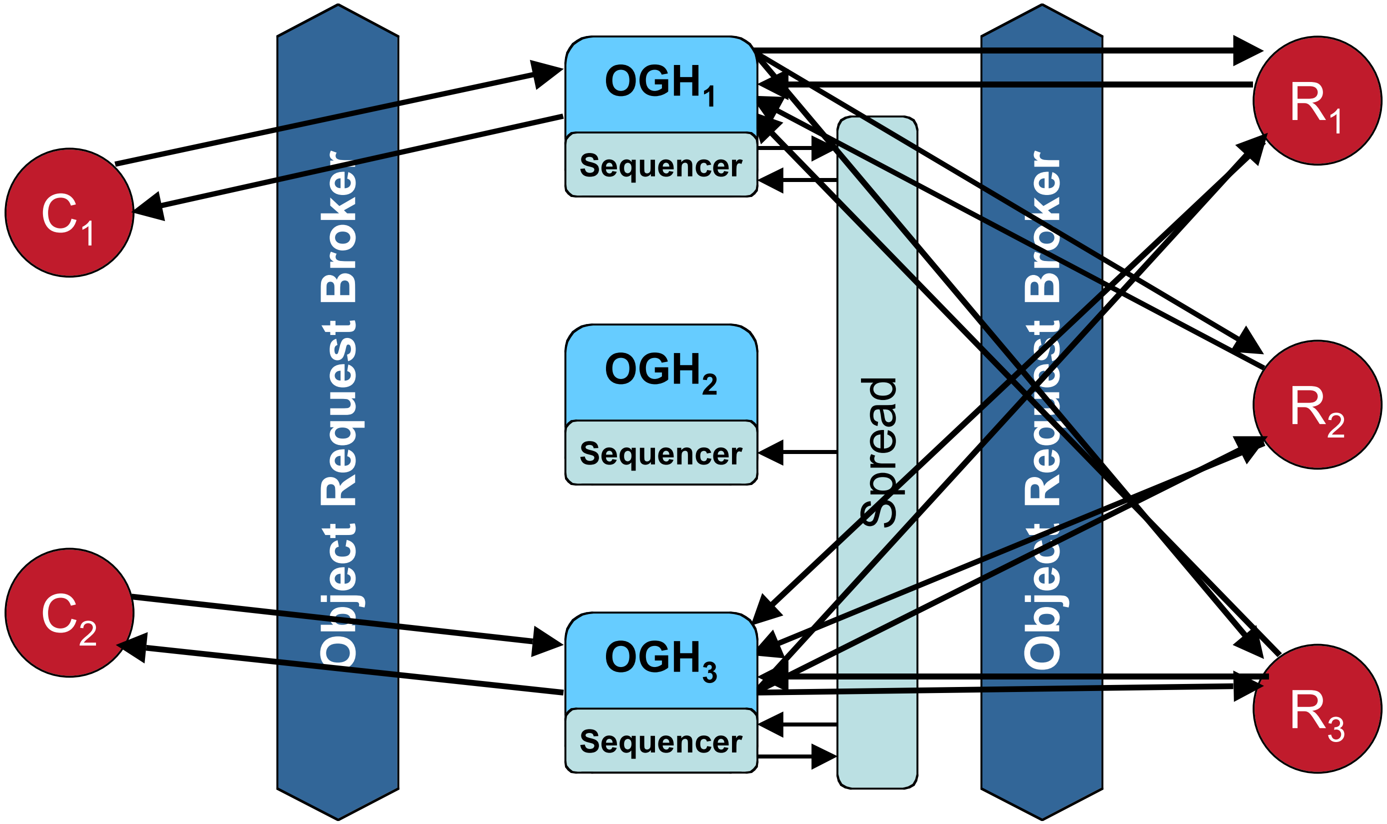
- Software infrastructure allowing to enhance the availability of deterministic CORBA objects
  - Test-bed platform for three-tier replication protocols
- Unifies benefits from CORBA and 3T replication
  - Interoperability
    - All the interactions among clients, middle-tier and replicas occur through IIOP
    - This allows interactions of CORBA objects running on different ORBs/OSs
  - Portability
    - Current version runs on Orbix, TAO, Orbacus
- Complies to the Fault-Tolerant CORBA standard
- [see also SRDS2002,SPE]



IRLArchitecture

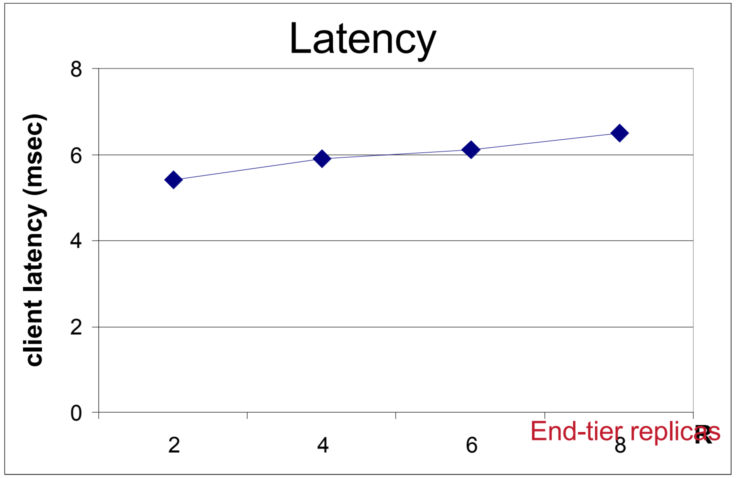


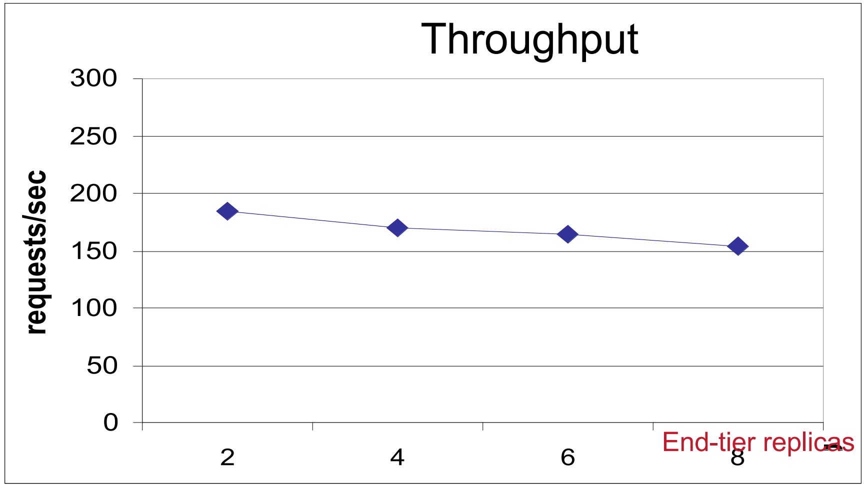
client/server Interaction in IRL



Performance

Prototype: C++, Orbacus, Maestro/Ensemble in the middle-tier to implement sequencer (uses hwmulticast in the middle-tier)





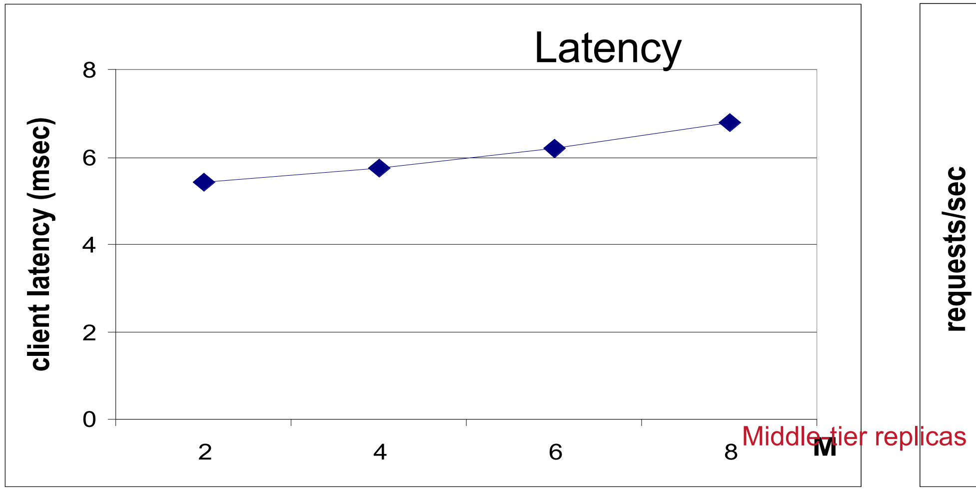
Setting:

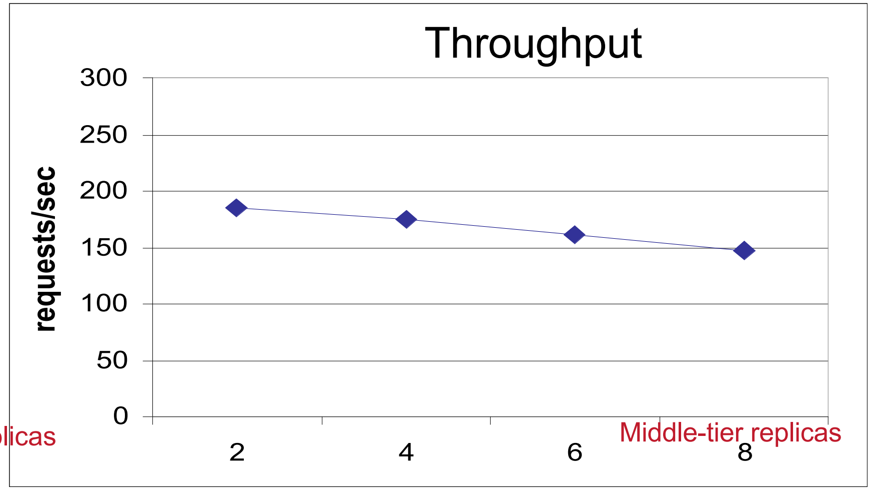
- 100Mbit SW-Ethernet LAN
- 4 clients
- 2 middle-tier replicas
- # of end-tier replicas varies from 2 to 8

Results

- Average latency increases by 3% per end-tier replica (5,5msec->6,5msec) using point-to-point (TCP) connections
- thput poorly reduces due to new end-tier replicas

Performance





Setting:

- 100Mbit SW-Ethernet LAN
- 4 clients
- # of middle-tier replicas varies from 2 to 8
- 2 end-tier replicas

Results

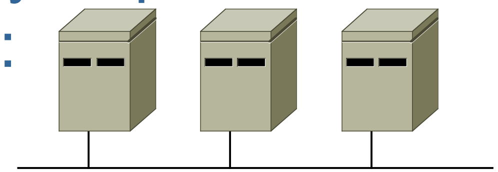
- Average latency increases by 3,5% per middle-tier replica (5,5msec->7 msec)
- Similar to the previous case due to the use of hw-multicast in the middle-tier

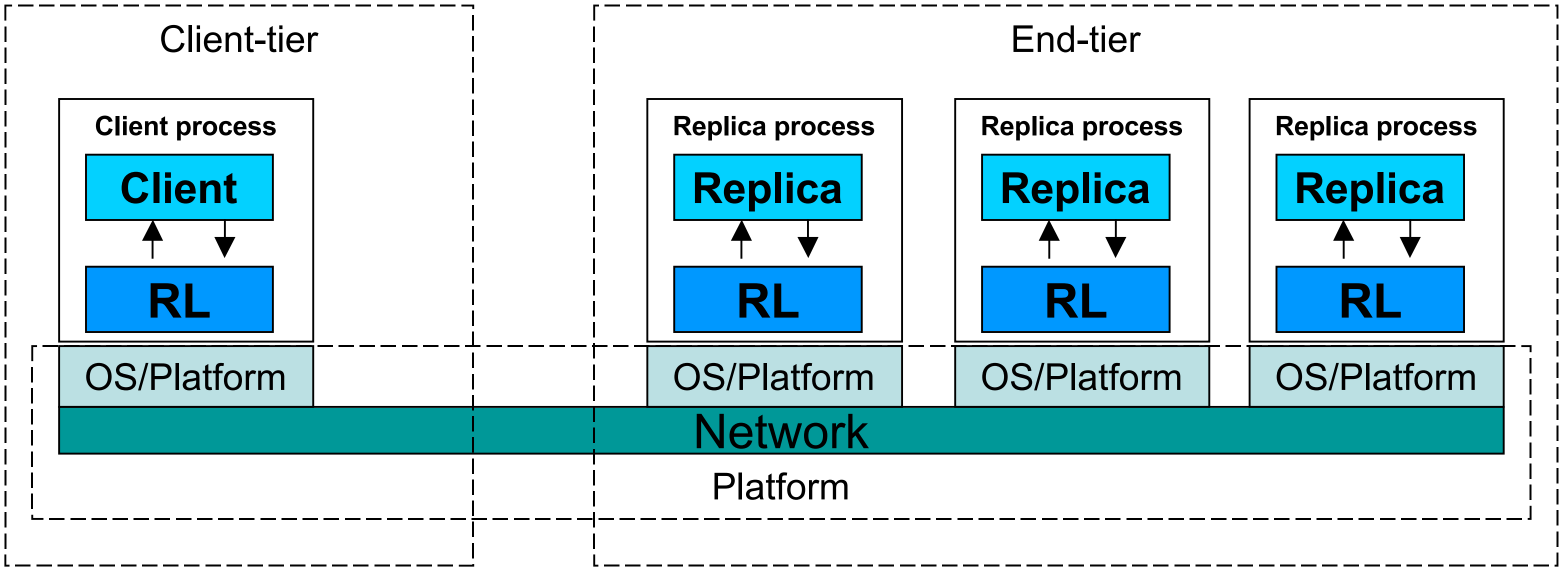
Concluding Remarks

- Three-tier replication
  - \+ Allows to deploy replicas in WANs
    - while enforcing strong replica consistency
    - avoiding delays due to replica asynchrony
  - \+ Confines in *Sequencer* the need for high coverage to get high availability
  - \+ Avoids replicas to run complex synchronization protocols, which can be costly in WANs
  - – Service availability bound to middle-tier
- The Interoperable Replication Logic
  - o Software infrastructure exploiting 3T replication to enhance CORBA objects with high availability
  - o Demonstrates feasibility of 3TAR
- Future and ongoing work
  - o Support of nondeterministic replicas
  - o Nested invocations
  - o Replication Management and dynamic groups

Implementation Issues

- Most of existing solutions to service high-availability with strong consistency requirements run over workstation clusters, i.e.:
  - a set of co-located workstation
  - interconnected by a local area network (LAN)
  - ensuring high coverage of partial synchrony assumptions
  - replication logic (RL) exploits group communications

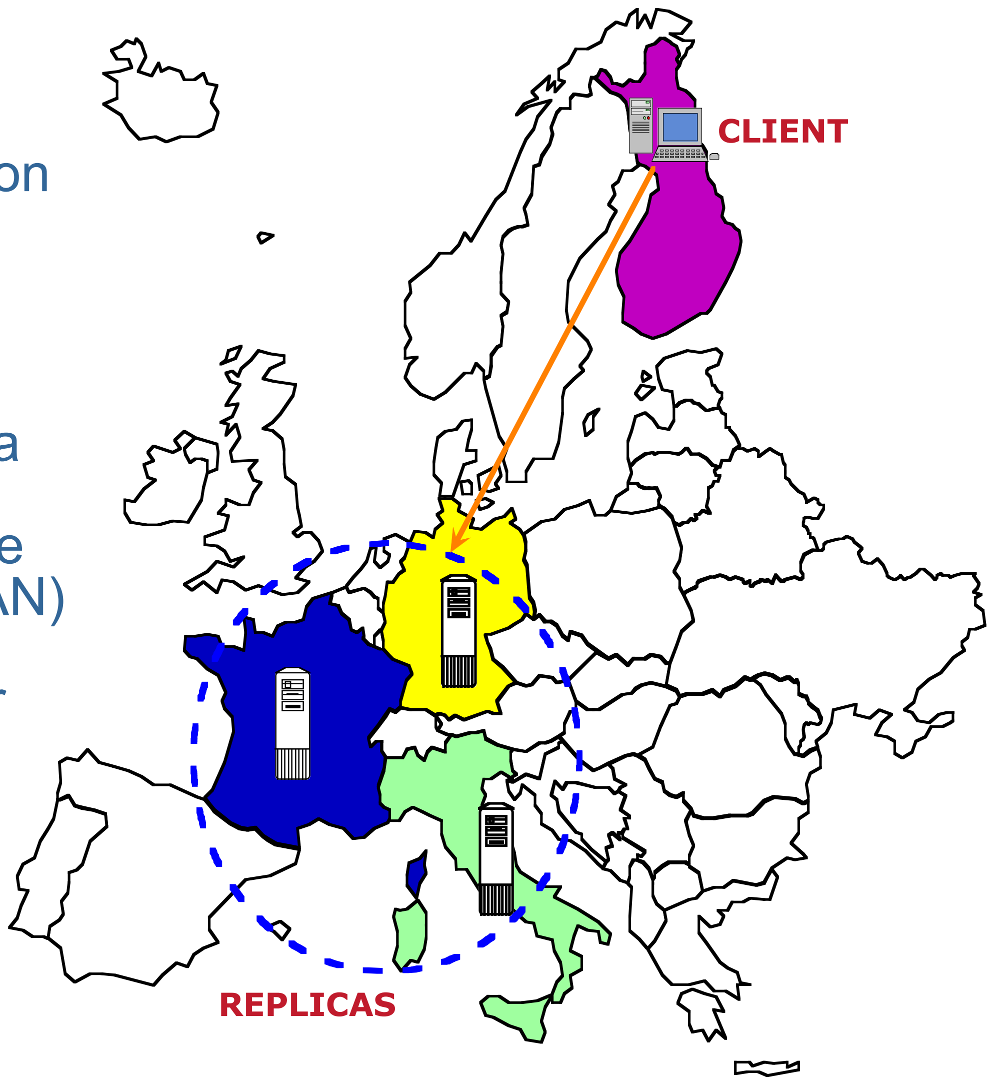




System Model

Message-passing asynchronous distributed system

- No upper bounds on message transfer and process scheduling delays
- Fits well our environment, *i.e.,* a set of nodes distributed on Wide Area Network (WAN) where we have unpredictable user and network load



Failure-free Run