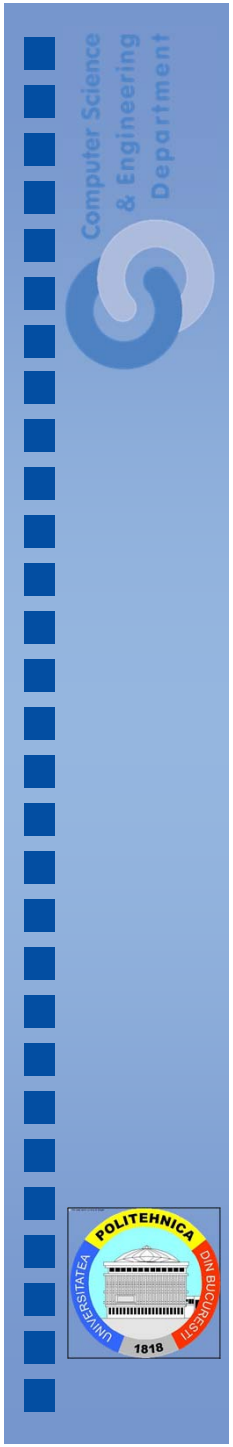


SYNASC 2009

Fault Tolerance using a Front-End Service for Large Scale Distributed Systems

Marieta Nastase, Ciprian Dobre, Florin Pop, Valentin Cristea

ciprian.dobre@cs.pub.ro

Faculty of Automatics and Computer Science
University POLITEHNICA of Bucharest
Romania

Outline

- A Service-Based Architecture
- Pilot Application
- Experimental Results
- Conclusions

Motivation

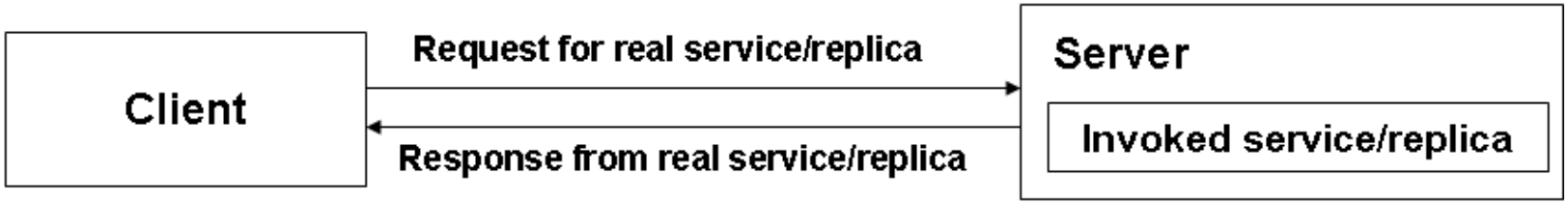
- Dependability of services used in large scale distributed systems → problem

- The proposed solution
  - → masks faults occurring in a set of replicated services by envisioning a container layer between them and clients.
  - → provides optimized access to the distributed services
  - → is specifically adequate to provide support to service providers
  - → do not rely on the use in the service development stage of any auxiliary functionalities or costs

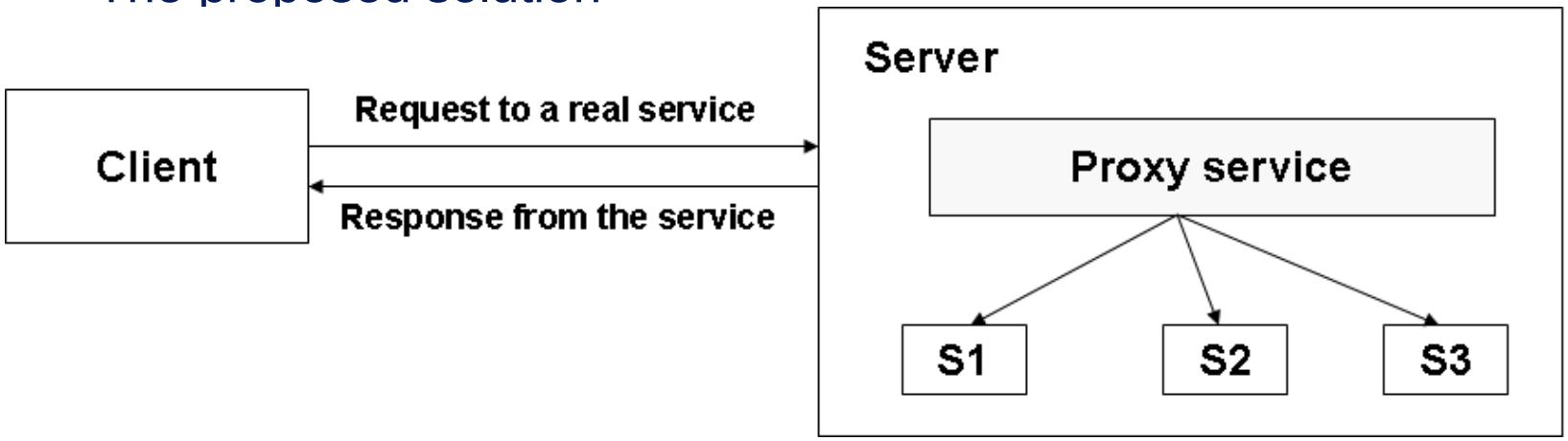
  - → does not require the alteration of the service or client → separation between the service implementation from the fault tolerance policy

The proposed solution

- A typical service invocation

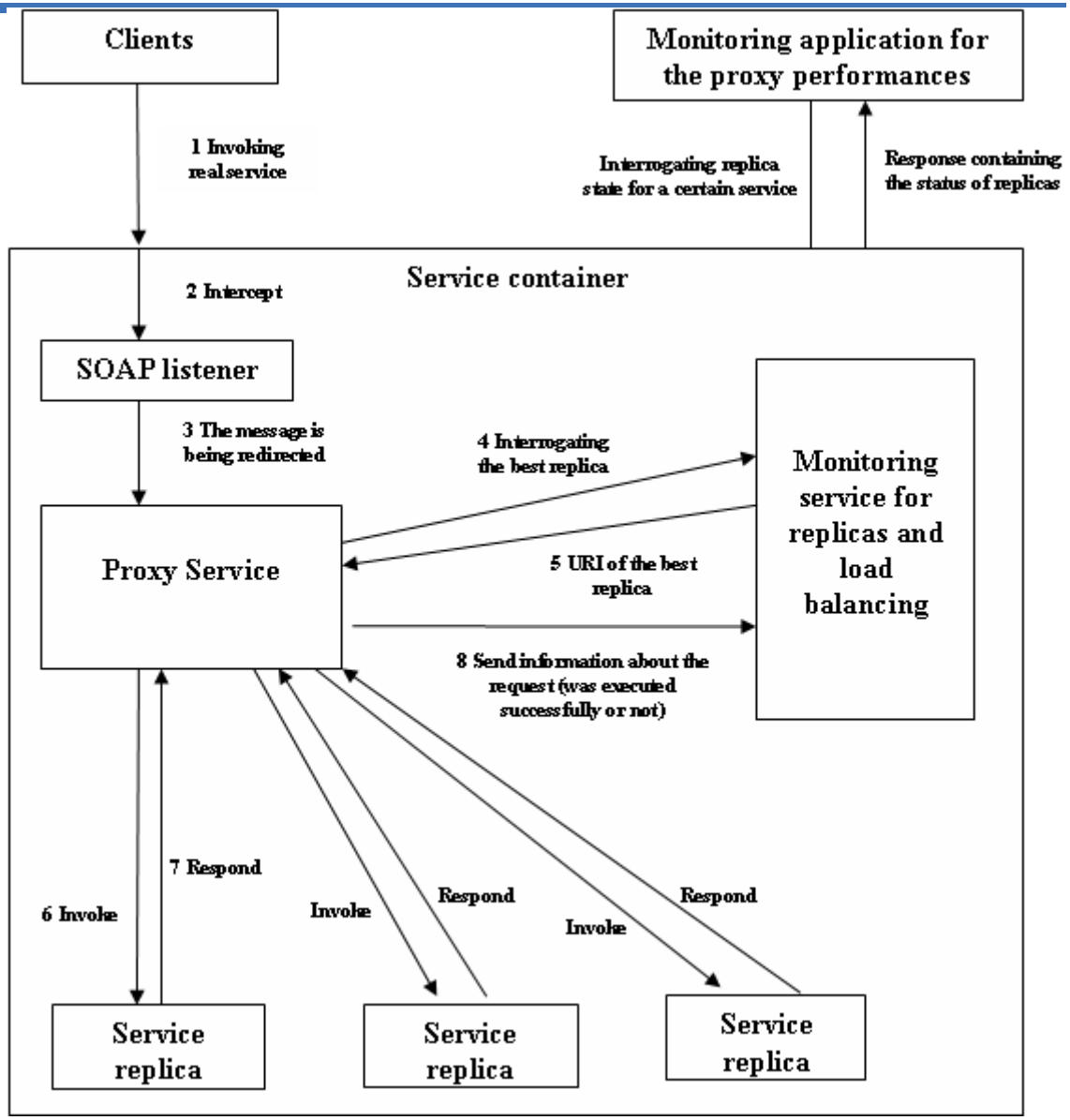


- The proposed solution



S1, S2, S3: replicas of the invoked service

The architecture (1)



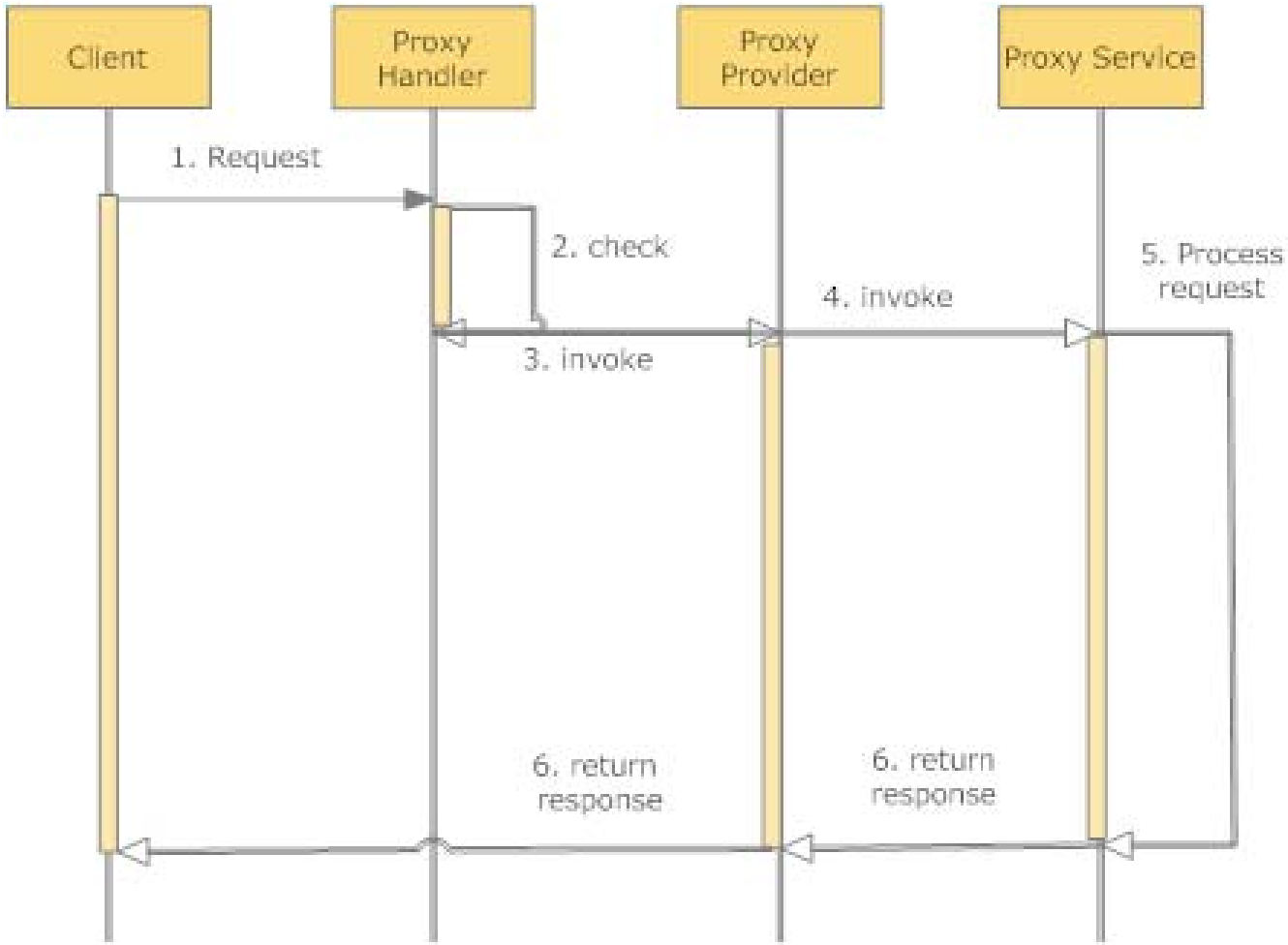
The architecture (2)

- **SOAP listener**
  - ➔ Monitors SOAP messages received by container
  - ➔ Redirects a message to the Proxy service or the real invoked service
- **Proxy Service**
  - ➔ Interrogates the monitoring service to find the address of the invoked service to which to redirect the request
  - ➔ Invokes the replica of the service
  - ➔ Handled errors by sending requests to other still-running replicas
- **Monitoring service**
  - ➔ Delivers data about the status of the replicas

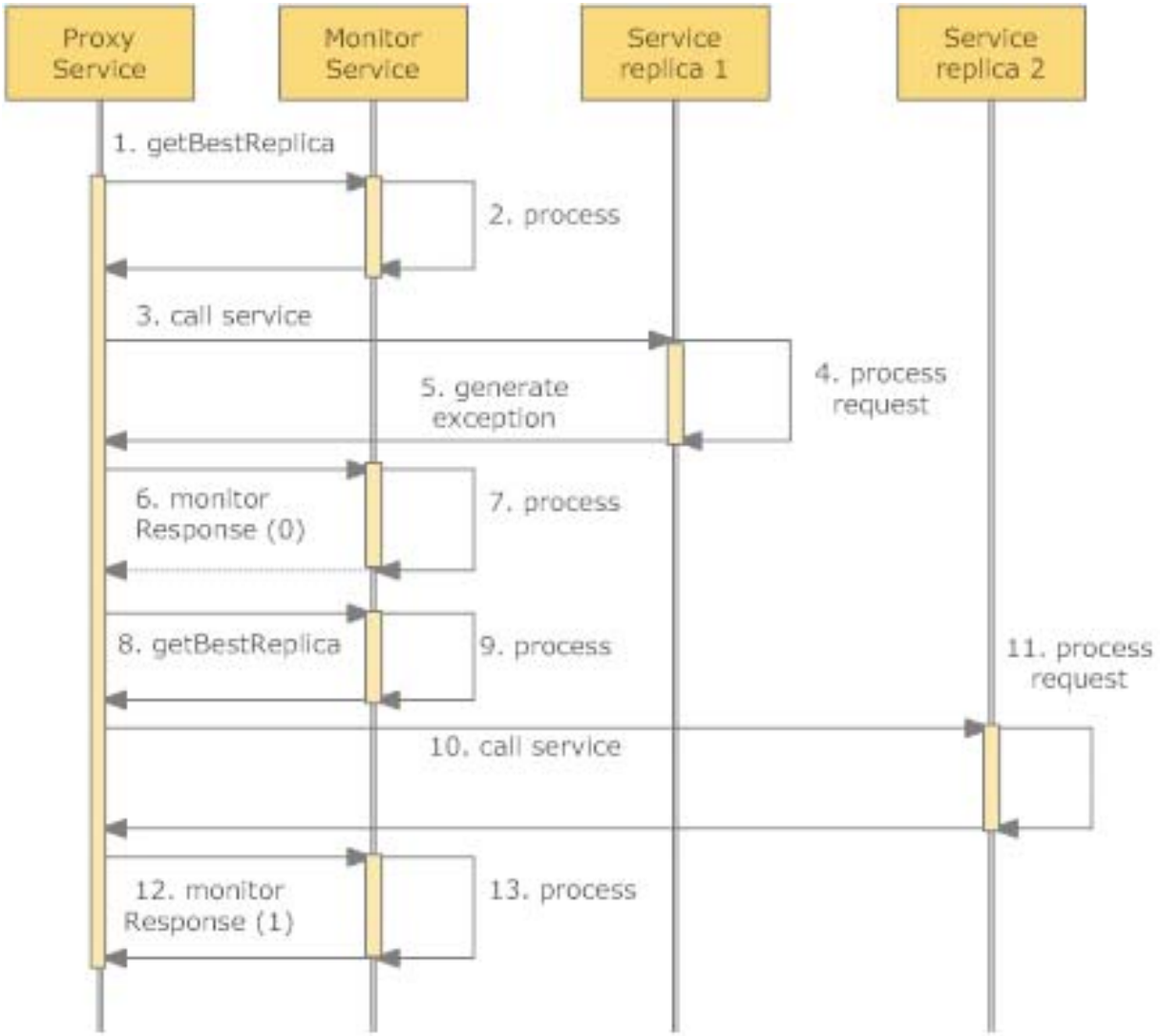
A pilot implementation

- In the AXIS architecture, we implemented the SOAP listener as a filter registered to the general request chain of Axis
  - ➔ can be configured to the deployment of the entire container or to just the deployment of certain services.
- The proxy service was implemented as a ”request” Web service, implemented inside an Axis container
  - ➔ lower response times
  - ➔ increased capability to process requests
- The proxy server also balances the load between service replicas
  - ➔ also considers the number of errors previously generated by a replica → mechanism for fault tolerance based on the history of failures

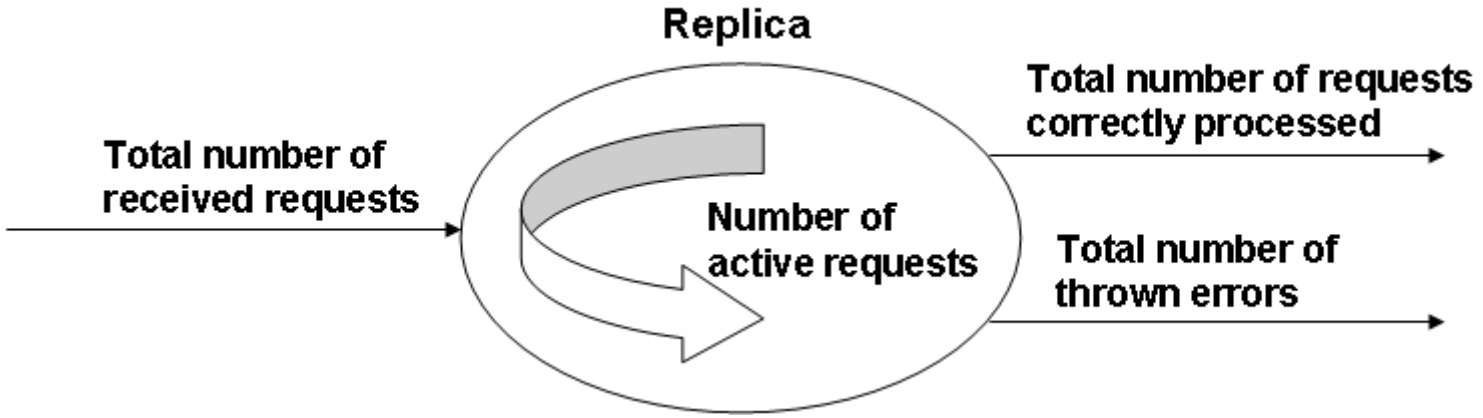
Message handling



The chain of actions executed by the Proxy service

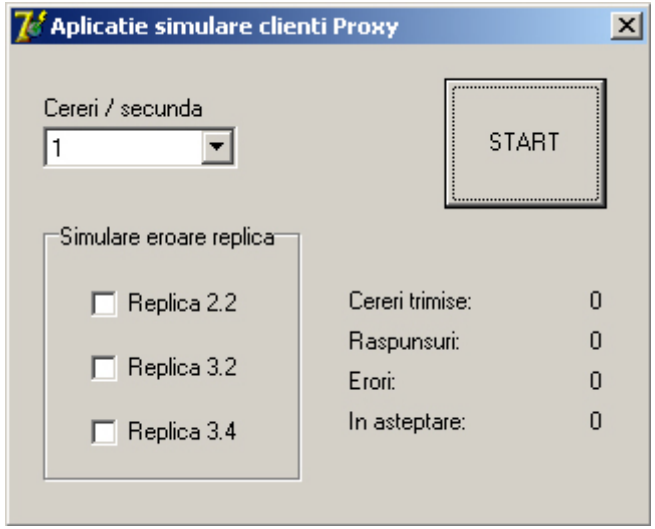


The information associated with each replica



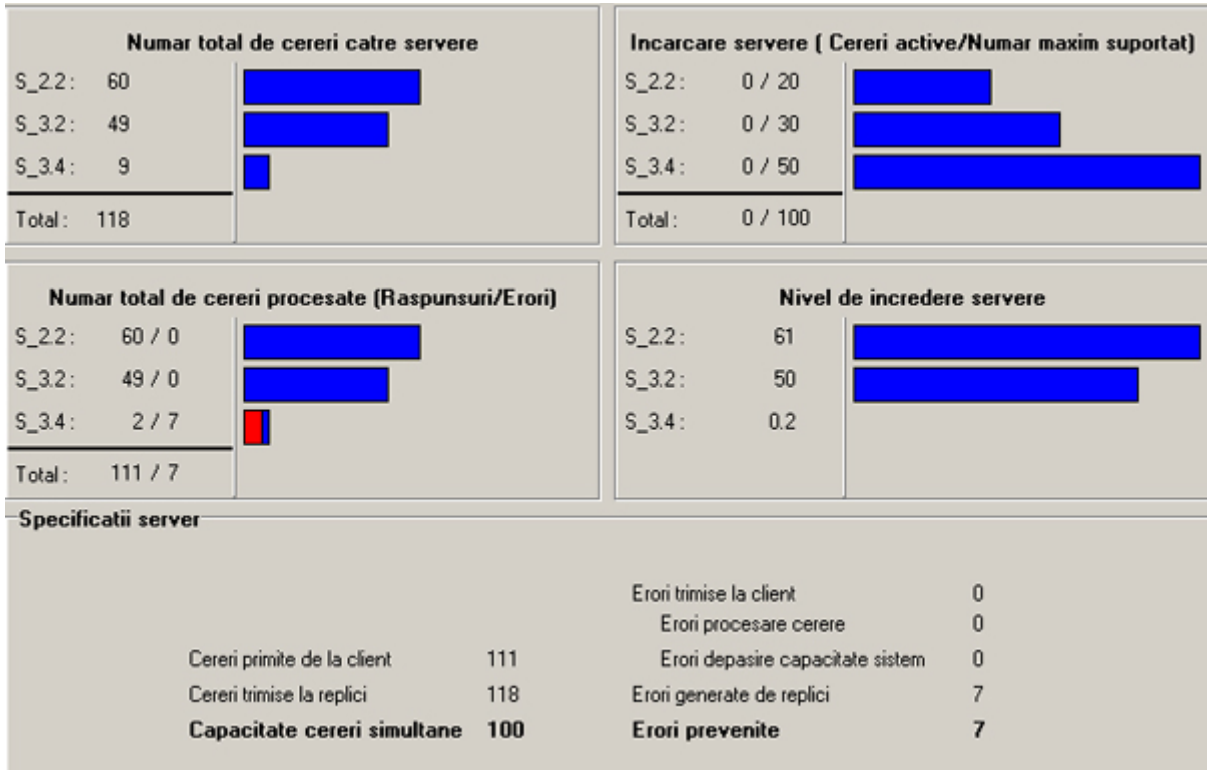
Experimental results (1)

- To evaluate the system we implemented and deployed a TestService service
  - ➔ simulates the behavior of a real computational service
  - ➔ serves a particular request in a certain amount of time
  - ➔ includes a mechanism to control if and when to trigger exceptions
- We also developed a client simulator, which generates requests for the TestService service

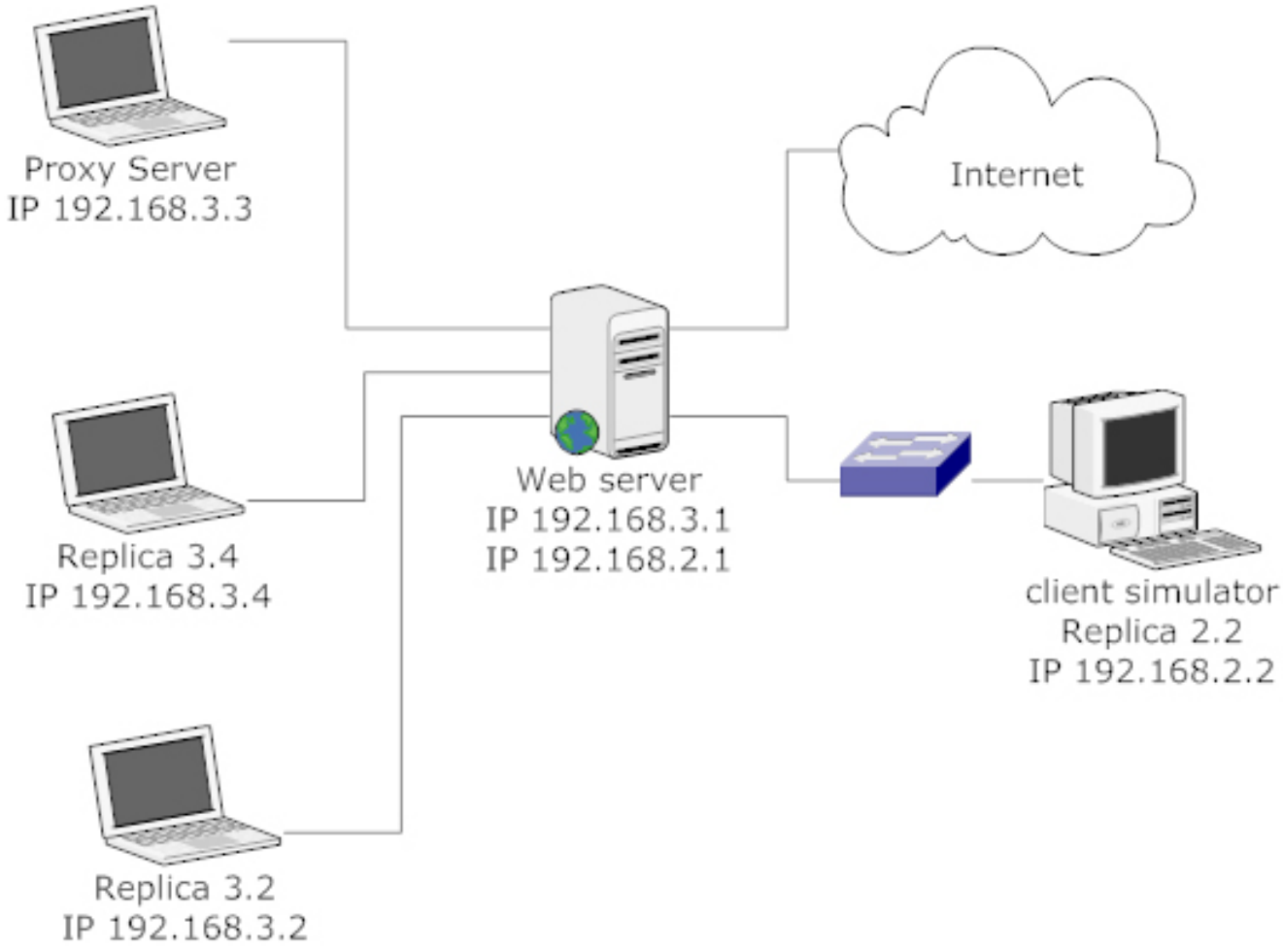


Experimental results (2)

- We also developed an application to monitor the performances of the system:



The topology used in the experiments



Experiments

- We conducted several evaluation experiments:
  - ➔ Evaluation of the possibility to mask possible failures
  - ➔ Simulation of high-load on the system because of the errors being generated by replicas
  - ➔ Evaluation of masking the overloading by using other replicas
  - ➔ Evaluation of increased capacity to mask processing errors

The evaluation of the possibility to mask possible failures

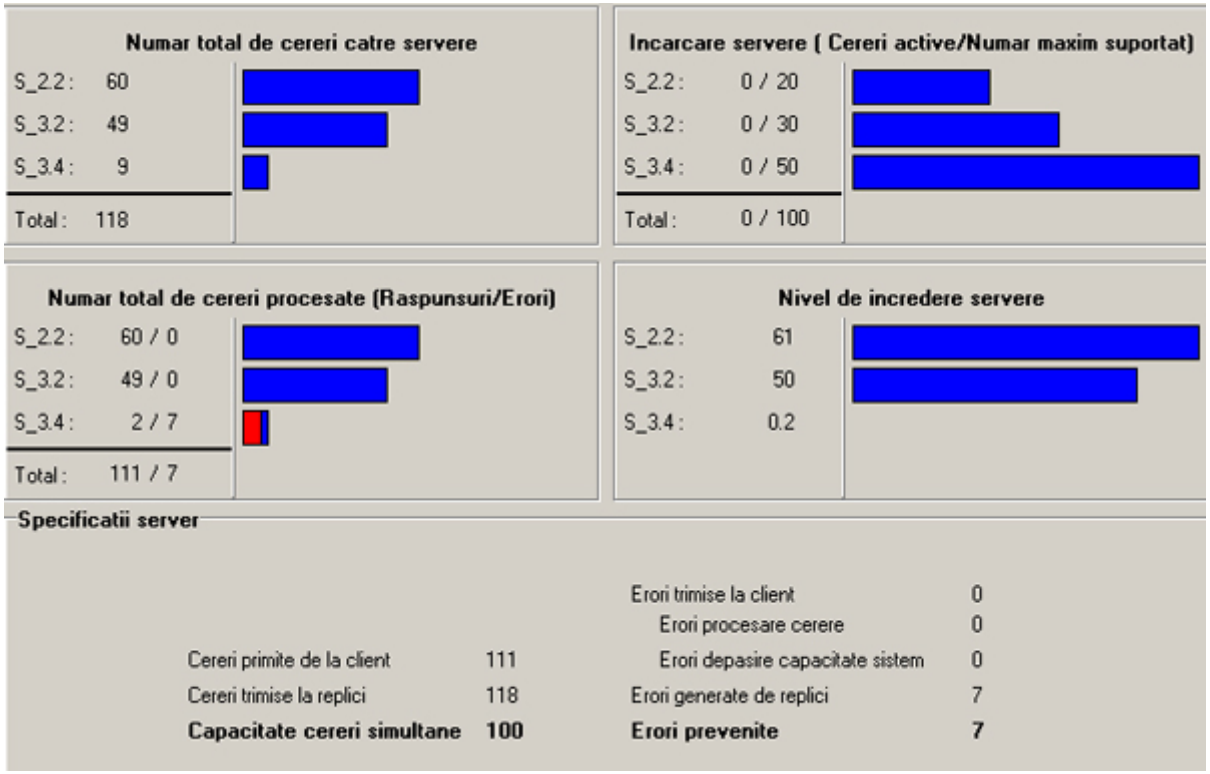
- In this experiment the load balancing algorithm behaved in a Round-robin approach
  - ➔ no replicas encountered errors
- Replicas were circularly and repeatedly selected to process each new incoming request
- The level of trust associated with each replica was 0.33
- At a certain moment the client set one replica to generate errors and then sent several more requests to be processed
  - ➔ the first requests were processed by replicas without any errors.
  - ➔ the next request resulted in an error being generated
- When the Proxy service received the error, it successfully retransmitted the request to another replica.

Evaluation of masking the overloading by using other replicas

- The client sends requests with a frequency of five requests per seconds. One replica was set to continuously generate errors.
- The number of requests exceeded the processing capacity of the two replicas working correctly → these requests were occasionally sent to the faulty replica
- That replica generated errors for all incoming requests and the Proxy service was then trying to retransmit them as a consequence to other replicas
- When all replicas became overloaded the Proxy service sent back the generated error to the client
- When at least one replica was not fully loaded with requests it could still serve the requests forwarded from the faulty replica and the client did not again see the generated error

System's Performance

- High capacity to mask processing errors
- For example, when one replica generates errors throughout the entire time of the experiment:

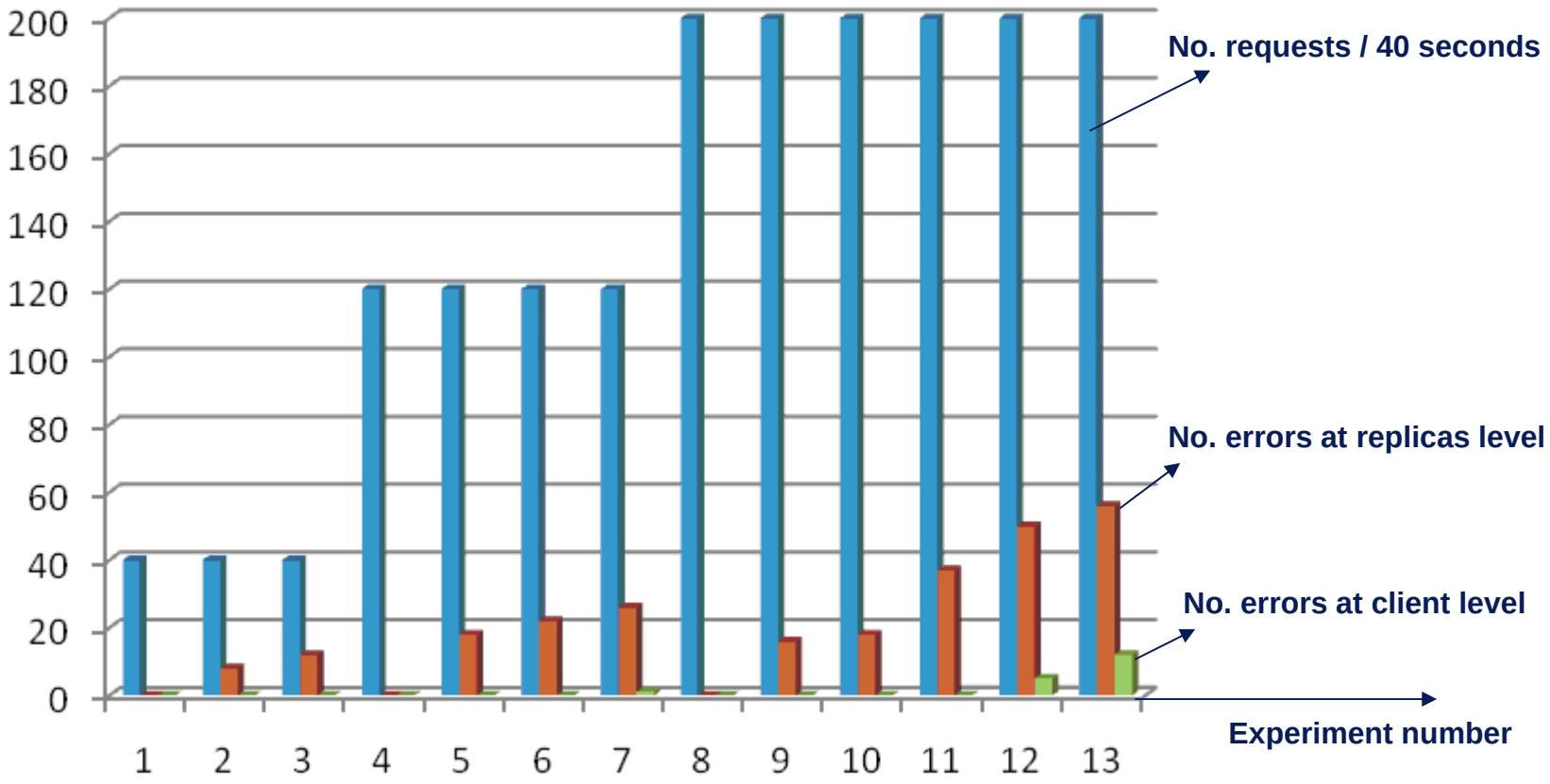


The results obtained in the experiment

Total number of requests sent by the client	The total number of requests processed by replicas	Number of errors generated by replicas	Number of errors as seen by the client
9	10	1	0
111	118	7	0
200	236	47	10
266	266	0	0
147	129	0	18

Experimental results

- We executed each experiment by varying the number of requests being sent per second
- The capacity to mask errors of the proxy system:



Conclusions

- Fault tolerance in large scale distributed systems by encapsulating replicated distributed services inside a container with the purpose of masking possible errors.
  - ➔ The client do not sees the error and the proxy service tries various automatic solutions to recover from occurred errors, such as forwarding the original requests to another non-faulty replica.
  - ➔ The solution uses a load-balancing system to ensure an efficient use of resources and the decrease of the response time.
  - ➔ The solution uses existing technologies that are already used by service providers → easily deployable in the context of both Web and Grid service.
- We demonstrated that the solution is viable and do not necessitate the addition of any supplementary mechanisms in the service implementation or the stub known by the client.
- In the future we aim to implement a replication mechanism for the Proxy service itself → single point of failure for the system

Questions?

Thank You!

florin.pop@cs.pub.ro