Performance Tradeoffs in Policies for Application Level Fault Tolerance

Theodoros Soldatos **Nantia Iakovidou**

Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
{niakovid, thsoldat}@csd.auth.gr

Abstract

Object oriented applications and services are composed of a number of objects with instances, which interact to accomplish common goals. Fault tolerance is attained via application transparent replication policies for masking faults that do not recur after recovery. Recently, we realized the advent of a number of middleware infrastructures and services, which allow customizing the replication characteristics of each one of the objects that comprise a system. In this paper we describe the most important performance tradeoffs we experienced in the frame of a simulation-based study, with regard to different fault detection conditions, different loss behavior cases for the faulty objects and alternative request-retry strategies. We point out the need for suitable quantitative design techniques, which will allow taking into account different concerns in a combined manner (e.g. fault tolerance combined with load balancing and multithreading). Each such technique also has to allow trading the gains of a potential change to the objects' replication properties against the imposed overhead, in order to fulfill the set design goal at the lowest possible cost.

Keywords:

Fault-tolerance, Software replication, Dependable systems

1. Introduction

Software replication is a well-known technique used to increase the availability of systems and services. Object replication has been recently standardized in a plain specification (OMG FT-CORBA in [8]) that describes how to apply operations in multiple replicas of an object in a transparent and consistent manner.

Systems' performance is affected not only by the applied replication policies but is also affected by the combined effect of a number of design choices such as what happens when a server object fails (loss scenarios), the applied request-retry policy which determines the object behavior when a response arrival is delayed, the applied request assignment to the available server objects, the applied multithreading scheduling algorithm etc. The interaction between all these factors has to be taken into account in terms of the caused simultaneous resource possession and the resulted hardware resource contention.

This difficulty is increased when the system is comprised of a number of interacting objects and the applied fault tolerance scheme is composed of miscellaneous replication policies for the constituent objects. This work investigates the performance tradeoffs that appear in this type of systems.

Object replication lies on the creation and management of multiple object replicas as a single object group. The client objects invoke methods on the server object group and one or more members of the server group execute the methods and return their responses to the clients, just like a conventional object.

In this context, object replication is used in minimizing loss of computation in the presence of non-recurrent faults. The role of a system or a service designer is to propose the architecture and the fault-tolerance policies for the provision of a possibly agreed level of quality of service (QoS).

Failures that do not reoccur after recovery may be caused by insufficient memory, media failures, power outages, non-partitioning network faults and the non-determinism introduced by distributed timers and the use of multithreading. We consider three distinct possibilities when a server object fails and two types of behavior for the period that a server object is unavailable. The three possibilities regarding the occurrence of an object fault are:

- all queued object requests are lost,
- only the request in service (if any) is lost and
- no request is lost.

The two types of behavior for the period that a server object is unavailable are:

- the server object continues to accept the object requests arriving while it is not available and
- the server object does not accept the incoming object requests while it is not available.

According to the OMG FT-CORBA specification we distinguish between the active (AR) and the passive replication (PR) styles.

In this paper, we make evident the performance tradeoff problems that appear in composite replication schemes. The experiments were performed by a combined reliability and system's traffic simulator and more precisely by the one described in [6].

The investigation of the described performance tradeoffs took place with regard to a case system study, where we had the chance to apply fault-tolerance schemes with different fault occurrence behaviors, request-retry timeout intervals, requests queue buffer sizes etc.

Section 2 outlines the functionality of the simulator. Section 3 describes the used case system model and its parameters. In section 4 we present and comment the results obtained with regard to the investigated performance tradeoff problems. Finally, the concluding section points out the need for suitable quantitative design techniques, which will

- allow the comparison of fault tolerance schemes that consist of different replication policies for the constituent objects and
- determine the optimal (lowest cost) fault tolerance configurations, with respect to precise dependability requirements.

2. The simulator and its performance evaluation approach

The used hybrid reliability and system's traffic simulator allowed us to take into account the interaction effects regarding:

- the simultaneous resource possession, caused by synchronous and often nested object invocations,
- the hardware resource contention, as a result of the replicas placement,
- the load and the blocking costs caused by the recurrent checkpointing/state transfer activities for the passively replicated objects (if any),
- the load caused by a replica restart and re-invocation of the logged requests (if any) and
- the delays due to re-invocation of requests lost, as a result of a (non-partitioning) network fault or as a consequence of a server object fault.

When an actively replicated object receives an invocation, every replica performs the operations in the same order and all the replicas receive the responses in the same order. This ordering of the operations ensures that the states of the replicas are consistent at the end of the operation.

For a passively replicated object there is a single designated replica, known as the primary that performs all the operations for the replicated object. The state of the primary and the sequence of the invoked methods are recorded in a log, according to the specified checkpoint properties. All the other object replicas are called backups and they do not issue or receive invocations and responses, while the primary replica is operational. Their sole purpose is to provide a pool of replicas from which a new primary can be chosen, if the current primary fails. In cold passive replication (CPR), the backup replicas are not even loaded into memory (activated) and thus they do not come into existence until the primary replica fails. In warm passive replication (WPR), the backup replicas are already created and initialized and the state of the primary is retrieved and transferred to all of the backup replicas in a frequency specified at the system configuration time.

The modeled fault detection mechanism assumes the existence of a transparent and fault tolerant fault monitoring service. Each object is periodically checked according to a specified time interval. Fault monitoring has been found to incur an approximate 5% increment, in the processor utilization, for about 500 milliseconds. This overhead has been taken into account.

The simulator supports request re-invocation in the following contexts:

- As a means utilized by a fault tolerance infrastructure or a client application to mask recipient faults and (non-partitioning) network faults, where the client-side does not detect the problem and receives no reply. The simulator allows the use of alternative request-retry timeouts and evaluates the accompanied overhead costs.
- Requests re-invocation also takes place in the course of a log-based recovery in a statefull object.

Both cases of requests re-invocation conform to the at-most-once invocation semantics of the CORBA object model, which means that each request is executed at most once. However, in the second case the recipient object replies with the result of the already invoked method, which incurs a certain processing cost for the server object. Duplicate message requests as a consequence of the employed replication scheme are detected and suppressed ([1], [2] and [7]).

3. The case system model

The used case system model is comprised of four (4) stateless service objects (instances of the class `SrvRequestAccepting`) and four (4) state owning objects (`obj1`, `obj2`, `obj3`, `obj4`) interacting as shown in Figure 1. Received class-1 and class-2 requests are assigned to the available service objects on a round-robin basis.

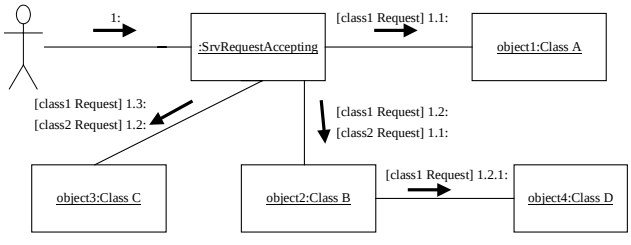

Figure1. Objects' collaboration diagram.

The four (4) stateless service objects as well as obj2, obj3 and obj4 are replicated according to the warm passive replication policy - with a single backup – described in [5]. Obj1 is actively replicated by utilizing two (2) object replicas.

The resulted composite fault tolerance scheme allows testing any combination of

- the scenarios mentioned in section 1 regarding the occurrence of an object fault and
- the two types of behavior also mentioned in section 1, for the period that a server object is unavailable,
  - with a request-no-retry policy or
  - alternative request-retry timeouts for the constituent objects.

Table 1 describes the initial roles and the placement of the sixteen object instances to the available process nodes. Collocated object replicas are processed in a processor sharing discipline and each of them is placed on a separate object server. Table 2 specifies the system model parameters used in all test cases.

Object instances placement:

process node 1	object 0 (backup)	object 1 (1st replica)	object 5 (primary)
process node 2	object 0 (primary)		object 5 (backup)
process node 3	object 2 (backup)		object 4 (backup)
process node 4	object 2 (primary)		object 4 (primary)
process node 5	object 3 (backup)		
process node 6	object 3 (primary)		
process node 7	object 6 (primary)	object 1 (2nd replica)	object 7 (backup)
process node 8	object 6 (backup)		object 7 (primary)

Table 1. Placement of objects' replicas.

object	state size (in KB)	no of invocations between checkoints/state transfers	fault process (exp. with rates)	state transfer speed (sec/KB)	class1 service (exp. with rates)	class2 service (exp. with rates)	reinvoked requests (exp. with rates)
0	-	no state	2r	-	0.05	0.05	-
1	0.9	AR	2r	0.8	0.52	-	0.1
2	0.7	60	r	0.6	0.25	0.25	0.1
3	0.5	30	r	0.6	0.7	0.7	0.1
4	0.6	90	2r	0.8	0.32	-	0.1
5	-	no state	2r	-	0.05	0.05	-
6	-	no state	2r	-	0.05	0.05	-
7	-	no state	2r	-	0.05	0.05	-
Object replicas restart times are exponential with rate 23.0 seconds.							
Both class 1 and class 2 requests arrivals are exponential with rate 2.5 seconds.							
Request assignment to the available service objects follows the Round Robin pattern.							
r = fault rarity = 21600 seconds							

Table 2. Model parameters used in all test cases

EXPERIMENTS		objects 0, 5, 6, 7	object 1	object 2	object 3	object 4
loss scenario	**I**	1	[2]	1	1	1
request-retry timeouts (in sec)		-	-	-	-	-
loss scenario	**II**	1	[2]	2	2	2
request-retry timeouts (in sec)		12	-	7.0	-	-
loss scenario	**III**	1	[2]	3	3	3
request-retry timeouts (in sec)		12.0	-	7.0	-	-
loss scenario	**IV**	1	[2]	1	1	1
request-retry timeouts (in sec)		12	-	7.0	-	-
loss scenario	**V**	1	[2]	1	1	1
request-retry timeouts (in sec)		14.0	-	9.0	-	-
loss scenario	**VI**	1	[2]	1	1	1
request-retry timeouts (in sec)		16.0	-	11.0	-	-
Fault monitoring intervals (in sec): 2.0, 4.0, 6.0, 8.0, 10.0,12.0						

Table 3. Varied model parameters.

Table 3 summarizes the tested combinations of fault-occurrence scenarios. The applied combination characterizes the fault-tolerance infrastructure used for the provision of the required level of dependability. The conducted experiments included the following three cases:

- **Loss scenario 1**: No request is lost on occurrence of an object fault and the object continues to accept incoming requests while it is not available.
- **Loss scenario 2:** All queued object requests are lost on occurrence of an object fault and the object does not accept the incoming object requests while it is not available. This is the default loss scenario for the object replicas of an actively replicated object.
- **Loss scenario 3:** No queued object requests are lost on occurrence of an object fault, but the object does not accept the incoming object requests while it is not available.

4. Performance tradeoffs and results

Fault tolerance performance in dependable object systems is ruled by a set of tradeoffs:

- Excessive checkpointing/state synchronizations result in performance degradation, since in the course of a checkpoint placement or a state transfer incoming requests cannot be processed before its end. On the other hand, deficient checkpointing and state synchronizations incur expensive recovery.
- Frequent request-retry timeouts and tight fault-monitoring intervals result in increased overhead costs, as opposed to infrequent ones, which may cause high requests response times.

In addition to the forenamed tradeoffs, fault tolerance performance also depends on the structural dependencies imposed by the objects' invocation flows. The used simulator utilizes appropriately designed performance measures, which allow trading the gains of a potential replication properties change, against the imposed overhead.

This is achieved by providing separate response time statistics for

- the service requests that are not affected by the occurred faults (fault unaffected) from
- the service requests that are affected by the occurred faults (fault-affected).

Fault-unaffected requests constitute the vast majority of the dispatched service requests, but response time guarantees in a dependable system also include the response times of the fault-affected requests.

The checkpoints placement tradeoff in dependable object systems has been already studied in [5] and [3]. In present paper, we provide interesting results regarding the effectiveness and the performance of the composite request-retry policies and loss scenario combinations shown in Table 3.

Figure 2 summarizes the performance results for the conducted experiments.

The fault tolerance scheme with a request-no-retry policy (SYSTEM I) does not exhibit significant overhead differences for the tested fault monitor intervals. However, we cannot use the request-no-retry policy for masking e.g. requests losses or queue buffer overflows in the server objects and non-partitioning network faults.

The fault tolerance scheme of SYSTEM IV is characterized by an extra overhead cost (as expected) and a steep increase of it, when using more effective fault-monitoring settings. This higher overhead is due to the continuously occurring timeouts in the senders, while the sent requests are queued in the recipients.

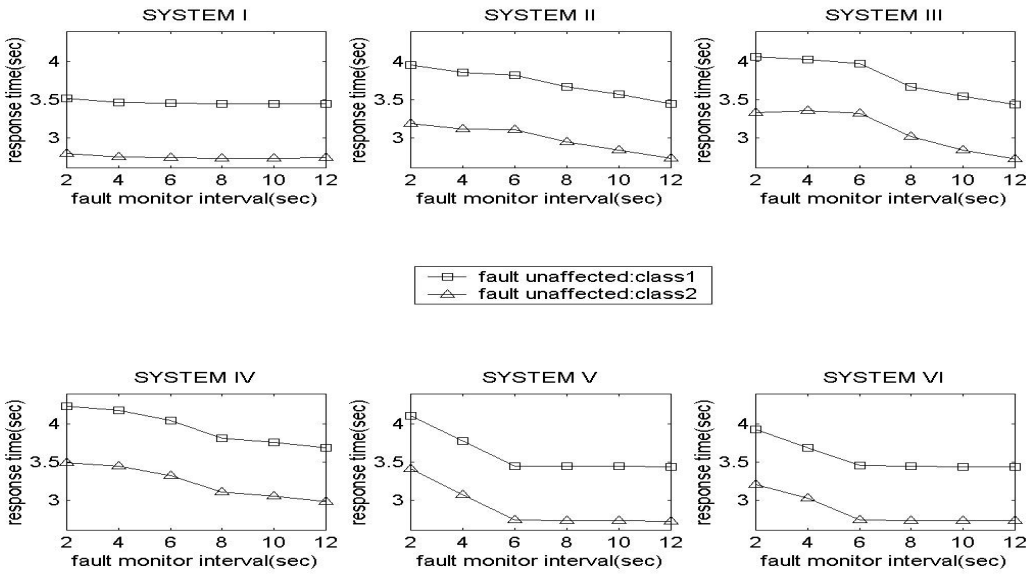

Figure2. Fault unaffected response times (sec).

When the already queued requests are lost at the time a server object fails and the incoming requests while the object is down are also lost (SYSTEM II), the overhead cost for the fault-unaffected requests is significantly lower. This comes as a consequence of the empty queues found by the (fault-unaffected) requests arriving in the just recovered operational primaries of the passively replicated objects.

When no queued requests are lost at the time a server object fails, but only incoming ones are lost (SYSTEM III), we obtain improved overhead costs compared to the case of SYSTEM IV and worse than in the case of SYSTEM II.

When the applied loss scenario is combined with less frequent request-retry timeouts (SYSTEM V & SYSTEM VI), we observe expanded possibilities to take advance of more effective fault detection settings, without additional overhead (fault-monitoring intervals from 12.0 to 6.0 sec). However, fault tolerance performance does not improve when increasing the used request-retry timeouts more than an appropriate threshold.

The selection of a composite request-retry policy with suitable timeout settings should be the subject of a systematic quantitative design method like the one introduced in [5] and [4].

Fault tolerance effectiveness is quantified by the means of the fault-affected requests shown in the graphs of Figure 2. All depicted means were obtained along with 95% confidence intervals with half width no more than 3% of the estimated value. We thus ensured statistically significant samples for the fault-affected requests (faults are by definition rare events).

Regarding the effectiveness of the tested configurations we draw the following conclusions:

- Excessively frequent request-retry timeouts do not improve fault-tolerance effectiveness due to the incurred overhead costs. This is shown when comparing the graphs of systems I, V and VI to the graphs of systems II, III, IV.
- The applied loss scenario has a slight impact on the resulted fault-tolerance effectiveness and this is obvious, when comparing the graphs of systems II and IV for the tightest fault monitoring intervals (from 2.0 and 4.0 sec).

5. Conclusions

In this paper we described the two most important performance tradeoffs that have to be taken into account in the design of dependable object systems. In such systems, the perceived performance and fault-tolerance effectiveness is also influenced by the complex structural dependencies imposed by the objects' invocation flows.

We provide interesting results with terms of a case system study. The experiments were performed by a combined reliability and system's traffic simulator and more precisely by the one described in [6]. The used simulator utilizes performance measures, fabricated especially for the support of suitable quantitative design techniques.

Such a quantitative design technique should

- make feasible the comparison of schemes, which consist of possible different replication policies and fault-tolerance settings,
- allow taking into account different design concerns in a combined manner (fault tolerance combined with load balancing and multithreading),
- be applicable in large-scale systems and

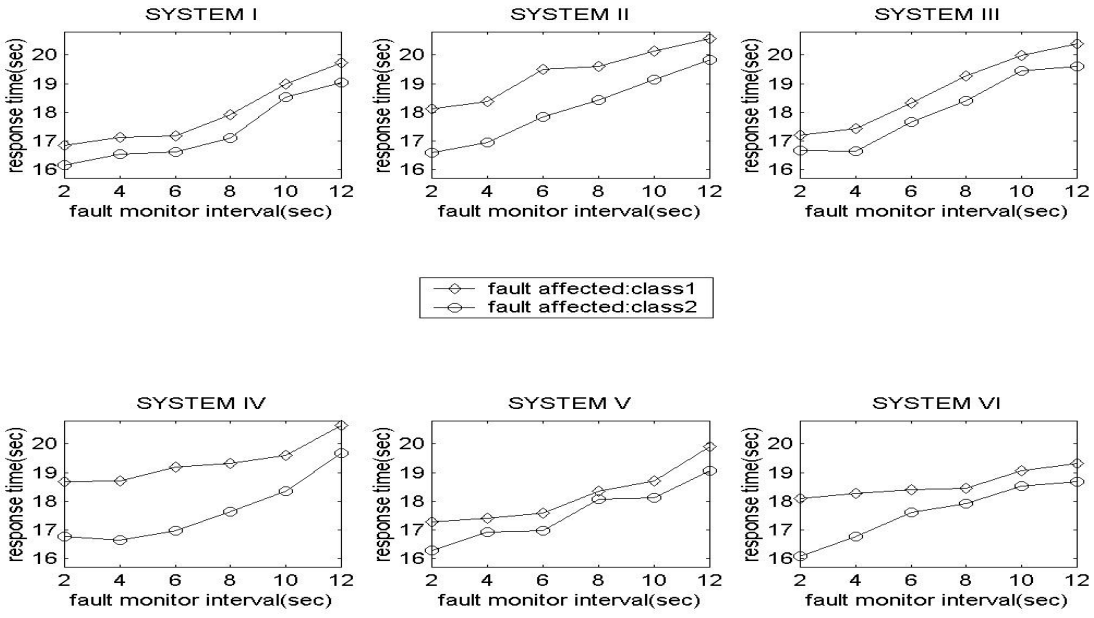

Figure 3.Fault affected response times (sec).

- allow determining optimal replication properties, with respect to precise response time guarantees.

6. References

[1] P. Felber, R. Guerraoui, A. Schiper, “Replication of CORBA Objects”, Distributed Systems, Lecture Notes in Computer Science 1752, Springer Verlag, pp. 254-276, 2000.

[2] R. Guerraoui, P. Eugster, P. Felber, B. Garbinato and K. Mazouni, “Experiences with object group systems”, Software: Practice & Experience, vol. 30, no. 12, pp. 1375-1404, 2000.

[3] P. Katsaros, L. Angelis and C. Lazos, “Simulation metamodeling for the design of reliable object based systems”, Proceedings of the EUROSIM 2004 Congress, EUROSIM, Paris, France, 2004.

[4] P. Katsaros, E. Angelis and C. Lazos, Applied multiresponse metamodeling for queuing network simulation experiments: problems and perspectives, In Proceedings of the EUROSIM 2001 Congress, EUROSIM, Delfts, The Netherlands, 2001

[5] P. Katsaros and C. Lazos, “Optimal object state transfer - recovery policies for fault tolerant distributed systems”, Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 04), IEEE Computer Society, Florence, Italy, pp. 762-771, 2004.

[6] P. Katsaros and C. Lazos, “A simulation test-bed for the design of dependable e-services”, WSEAS Transactions on Computers, WSEAS Press, Vol. 4/2, pp. 915-919, 2003.

[7] P. Narasimhan, L. E. Moser and P. M. Melliar Smith, “Strong replica consistency for fault-tolerant CORBA applications”, Journal of Computer Systems Science and Engineering, CRL Publishing, 2002.

[8] Object Management Group, Fault tolerant CORBA, OMG Technical Committee Document, 2001-09-29, September 2001.