SIMULATION METAMODELING FOR THE DESIGN OF RELIABLE OBJECT BASED SYSTEMS

Panagiotis Katsaros Lefteris Angelis Constantine Lazos

Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
{katsaros, lef, clazos}@csd.auth.gr
http://delab.csd.auth.gr/~katsaros/index.html

ABSTRACT

Replication is a suitable approach for the provision of fault tolerance and load balancing in distributed systems. Object replication takes place on the basis of well-designed interaction protocols that preserve object state consistency in an application transparent manner. The published analytic performance models may only be applied in single-server process replication schemes and are not suitable for schemes composed of miscellaneous policies, such as those arising in object based systems. In this work we make use of a simulation metamodeling approach that allows the comparative evaluation of composite fault tolerance schemes, on the basis of small size uniform experimental designs. Our approach opens the possibility to take into account different design concerns in a combined manner (e.g. fault tolerance combined with load balancing and multithreading). We provide results in terms of a case system study that reveals a dependence of the optimal adjustments on the system load level. This finding suggests the device of dynamically adjusted fault tolerance schemes.

KEYWORDS

object replication, fault tolerance, simulation metamodeling

1 INTRODUCTION

Distributed object applications and services are composed of a number of objects with instances that interact to accomplish common goals. When designing and deploying such an application there are many decisions to be made with regard to the composition of objects into processes, the distribution of processes across nodes, the threading policies for the processes and the appropriate software replication for the provision of the required levels of fault tolerance and/or load balancing. All these decisions affect the resulted quality of service (QoS).

In this paper we provide a quantitative technique for the selection of a composite replication scheme that assures the required fault tolerance effectiveness. A scheme's effectiveness is determined by the response times of the fault-affected service requests, which often have to conform to a specified service quality level. The proposed simulation-based evaluation takes place on the basis of a simulation metamodeling approach that allows us to take into account different design concerns in a combined manner (e.g. fault tolerance combined with load balancing and multithreading). Metamodeling is used for the prediction of system performance and its optimum operating conditions.

In the design of fault tolerance, different candidate policies may be considered and their optimum operating conditions is the unique criterion making feasible their comparison. The reason lies in the mechanisms used in replication protocols for minimizing loss of computation in the presence of faults.

One such mechanism well known as checkpointing saves the objects' states from time to time on stable storage. Another mechanism is the object state transfer (synchronization) of a live or recovered object to the states of the other replicas. In both cases the participating objects have to be operationally quiescent, i.e. not to be in-between an invocation service or to be blocked, because of a synchronous invocation of another object. In the course of a state transfer, the received invocations cannot be processed, before the end of it. Thus, placement of checkpoints and state transfers has a major impact on the perceived quality of service: excessive checkpointing (state transfers) result in performance degradation, while deficient checkpointing (state transfers) incurs expensive recovery. Moreover, in schemes composed of multiple policies for the constituent objects, checkpointing performance also depends on the structural dependencies imposed by the objects' invocation flows.

In [6] we used the term "tightest effective checkpoint intervals" for the checkpoint (state transfer) placement that yields a scheme's optimum effectiveness. We also provided a heuristic decision-making procedure that converges to the forenamed checkpoint placement. This procedure takes advantage of an initially unknown number of simulation runs that depend on the number of interacting objects. Its efficiency is bound to the observed effectiveness variations, because it trades the gains of a checkpoint interval reduction, against the overhead imposed to the vast majority of service requests (which are not affected by the occurred faults). This trade-off is suitable for the choice of a minimal cost checkpoint placement, in respect to a specified service quality level. However, it is not an efficient way for determining the required tightest effective intervals.

In the present paper this problem is bypassed by the use of small size uniform experimental designs with an a priori known number of runs.

In related work we can refer only to the work published in [8], where the authors propose a hybrid mathematical programming and analytic evaluation algorithm for a different trade-off problem: to determine process replication or threading levels, such as to avoid unnecessary queuing delays for callers or unnecessary high consumption of memory. Their approach while more efficient compared to simulation metamodeling, does not allow taking into account different design concerns in a combined manner (e.g. fault tolerance combined with load balancing and multithreading).

On the other side, the most recent work regarding the estimation of the optimal checkpoint interval is the one published in [2]. This work concerns with minimizing the program execution time and maximizing the effectiveness in a single node in a mobile environment with handoffs. The whole approach fits to a different computational environment and as other analytic models is also not suitable for the evaluation of schemes composed of miscellaneous policies.

Section 2 constitutes a short introduction to object-based fault tolerance. Section 3 outlines the core evaluation approach. Section 4 introduces the used case system study and summarizes the obtained results. Finally, the paper concludes with a discussion on future research prospects.

2 FAULT TOLERANCE IN OBJECT BASED SYSTEMS

Fault tolerance for object-based systems has been recently standardized [10] in a plain specification (OMG FT-CORBA) of robust support for applications that require a high level of reliability. To render an object fault tolerant, several replicas of the object are created and managed as a single object group. The OMG FT-CORBA standard allows the definition of appropriate fault tolerance properties, for each constituent object group. The supported strategies include request retry, redirection to an alternative server, passive (primary/backup) replication and active replication.

The provided support [9, 4, 11] offers protection against object faults that do not recur after recovery. Some of them may be hardware dependent (e.g. insufficient memory) and others may be attributed to media failures, power outages, human lapses, catastrophic events, the use of local timers, the use of multithreading etc. The faults conform to the fail-stop model, which means that objects fail by crashing, without emission of spurious messages. There are not any assumptions about the network topology or the protocols making up the interprocess communication service, except that communication is accomplished through loss less FIFO channels. Network partitioning and commission faults are not addressed.

In active replication all the group replicas execute each request independently, but in the same order. Checkpointing on a regular basis is not required. In passive replication only one object replica - the primary - executes the methods invoked on the group. Checkpointing takes place on a regular basis. In the presence of a fault, a backup replica is promoted to be the new primary. The state of the new primary is restored to the last checkpointed state of the old primary and the requests logged since the last saved checkpoint are then reapplied.

A fault tolerance scheme for an object-based system may be composed of miscellaneous policies. The term miscellaneous refers to the replication styles for the objects that comprise the system, with possibly different fault tolerance properties (e.g. different checkpoint placement). The simulator described in [5] allows us to realistically model the interaction effects regarding:

- the simultaneous resource possession, caused by the synchronous, often nested object invocations, which block the callers, until they get a reply,
- the hardware resource contention, as a result of the chosen replica placement,
- the load and the blocking costs caused by the recurrent checkpointing activities and the state transfers between replicas of the same group and
- the load, caused by a replica restart (repair) or re-invocation of the logged requests, according to the OMG FT-CORBA specification [10].

3 THE CORE EVALUATION APPROACH

The core evaluation approach allows the comparison of (combined) fault tolerance (and load balancing) schemes, with at least one recurrent checkpointing activity. A scheme's effectiveness is measured by the mean of the fault-affected requests' response times. We consider a service request to be affected by the occurred faults if

- its dispatch to the assigned service object is delayed or
- its dispatch causes the generation of synchronous invocations that are queued somewhere in the system,

as a result of at least one detected object fault. Regarding actively replicated objects a synchronous invocation is considered to be affected by an object fault, if it is delayed because of blocking to enforce operational quiescence and/or a state transfer for an object replica recovery.

More frequent checkpoints are considered to be effective, when they result in a reduction of the fault-affected requests' response times. If there is no chance of further improvement for all possible interval reductions in a vector of n intervals - where n the number of objects with a recurrent checkpointing activity -, this vector specifies the tightest effective checkpoint intervals.

For a (combined) fault tolerance (and load balancing) scheme, the tightest effective intervals determine the minimum response times that the scheme may achieve for the fault-affected service requests. We say that this vector characterizes the scheme's optimum effectiveness for the applied object fault model. These checkpoint intervals constitute the unique criterion that makes feasible the comparison with other schemes composed of possibly different replication policies, checkpoint placement mechanisms and/or load balancing strategies.

The tightest effective intervals were found by metamodeling [7] based on a small size uniform experimental design with an a priori known number of runs. The uniform experimental designs [3] have been suggested for computer and industrial experiments specifically for cases where the underlying relationships are unknown. They are space-filling designs with experimental points scattered uniformly on the domain. They can explore complicated nonlinear relationships between the response variable and the factors with a reasonable number of runs and have been proved robust to the underlying model specifications. A great number of them has been tabulated in [12], where we also selected the design used in our work.

The checkpoint intervals that were found to be significant were set to the values minimizing the fault-affected response times. For the non-significant ones or when multiple minima were detected, the values minimizing the fault-unaffected response times (lowest cost fault tolerance) were selected.

4 A CASE SYSTEM STUDY

The system model used (Figure 1) in the case study is comprised of four (4) interacting state owning objects (`obj1`, `obj2`, `obj3`, `obj4`) and four (4) stateless service objects (instances of the class `SrvRequestAccepting`). Received class-1 and class-2 requests are assigned to the available service objects in a random probabilistic fashion with equal probabilities.

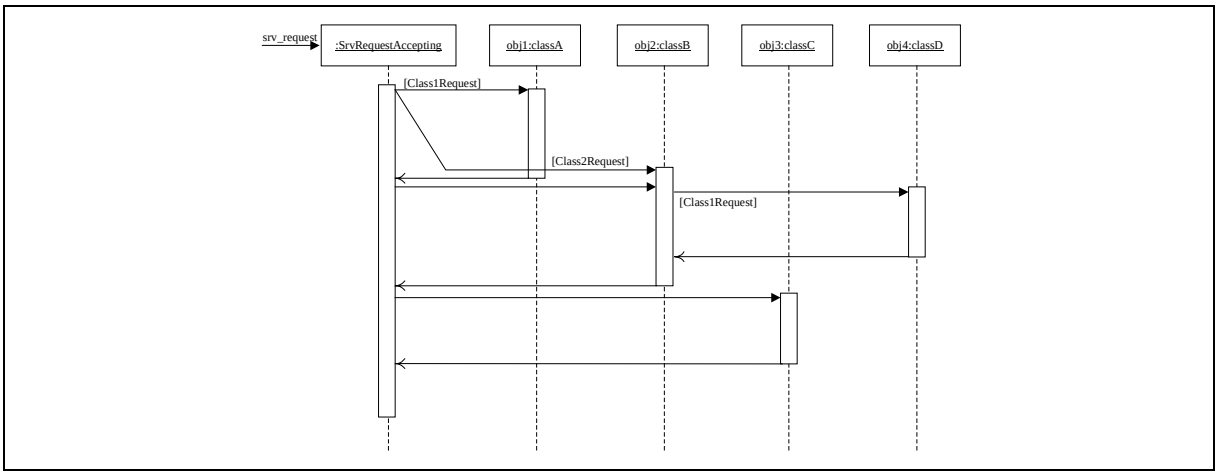
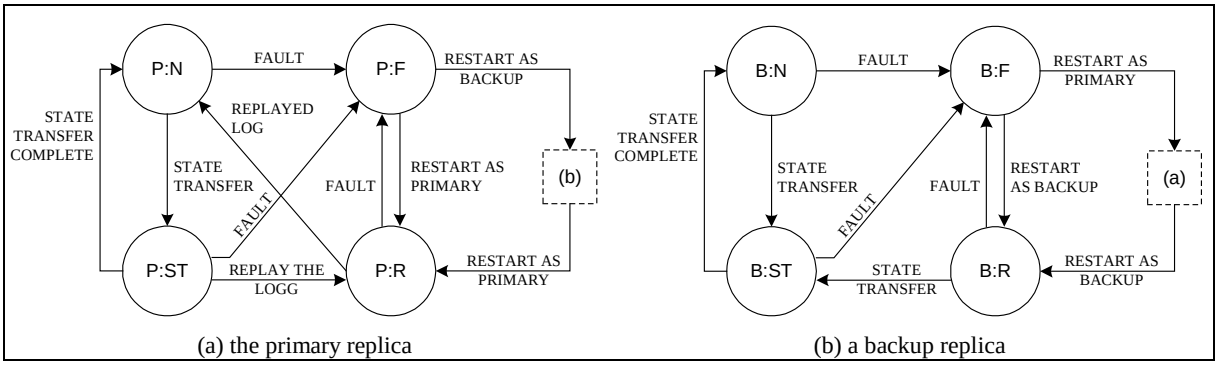


Figure 1. Message sequence for the objects of the case system



(a) the primary replica (b) a backup replica

Figure 2. Passive replication with one primary and one backup object per group

Objects are replicated according to the passive policy of Figure 2, with one primary and one backup per object group. We consider the concurrent execution of the two replicas that alternate in the primary and the backup roles, as shown respectively in the state transitions of (a) and (b). State transfers (for the state owning objects) correspond to backup state updates for a live backup and always result in a persistent checkpoint. They take place on a regular basis according to the specified checkpoint placement. Thus the applied composite schemes result in checkpoint placements specified as quadruples. Generally, in any composite scheme, only the constituent policies that are based on regular checkpointing contribute to the number of experiment's factors. This justifies the applicability of the proposed tightest effective intervals based evaluation in real size applications and services.

State transfer transitions (P:N→P:ST, B:R→B:ST) are also used as a state update of a recovered backup to make feasible a potential replacement of the primary in case of fault. When a failed primary is detected, the corresponding backup replaces it and a replica restart is then scheduled to occur (P:F→(b)) for the new backup. A recovering primary (P:R) either restarts or executes the invoked requests having logged since the last checkpoint. A recovering backup (B:R) either restarts or waits for a state transfer (B:R→B:ST), to become operational (B:N). Each replica restart restores it to the last checkpointed object state. According to the OMG FT-CORBA specification [10], re-invoked requests are detected and the same operations are not performed more than once.

The state transitions of Figure 2 are used in two distinct fault tolerance schemes:

- one with load-dependent checkpoint intervals (LDSC) and
- another one with periodic checkpoint intervals (PSC),

for the objects, which own state. The first scheme assumes a specified number of serviced invocations between checkpoints and the second one results in a fixed time interval between them. Resource consumption for the performed state transfers depends on the object state sizes and the required CPU time (state transfer speeds in sec/KB).

The four (4) service objects (`obj0, obj5, obj6, obj7`) are placed to the available nodes as specified in Table 1. Computational resource consumption is exponentially distributed both for the invoked requests and for the performed state transfers. Collocated object replicas are processed in a processor sharing discipline and each of them is placed on a separate object server. Finally, Table 1 summarizes the used parametric object fault model.

Table 1. System model parameters

service objects:	objX:SrvRequestAccepting (X=0, 5, 6, 7)			
load balancing:	random probabilistic with equal probabilities of request assignment to service object X			
class 1 request arrivals:	exponential with rates (sec)	2.6	2.4	2.2
class 2 request arrivals:	exponential with rates (sec)	2.6	2.4	2.2

	obj1:classA	obj2:classB	obj3:classC	obj4:classD
object state size (KB):	0.9	1.1	0.8	0.6

object replicas:	rep10 obj1	rep11 obj1	rep20 obj2	rep21 obj2	rep30 obj3	rep31 obj3	rep40 obj4	rep41 obj4	repX0 objX	repX1 objX
class 1 service (exp.)	0.52	0.57	0.6	0.6	0.83	0.83	0.32	0.32	0.2	0.2
class 2 service (exp.)	-	-	0.28	0.28	0.83	0.83	-	-	0.2	0.2
reinvoked requests (exp.)	-	-	-	-	-	-	0.1	0.1	-	-
state transfer speed -sec/KB (exp.)	0.8	0.8	0.6	0.6	0.6	0.6	0.8	0.8	-	-

object replicas placement:				
process node 1	rep00 (obj0)	rep51 (obj5)		(stateless) service object
process node 2	rep01 (obj0)	rep50 (obj5)		(stateless) service object
process node 3	rep11 (obj1)	rep21 (obj2)	rep40 (obj4)	state owning object
process node 4	rep10 (obj1)	rep20 (obj2)	rep41 (obj4)	state owning object
process node 5	rep30 (obj3)			state owning object
process node 6	rep31 (obj3)			state owning object
process node 7	rep60 (obj6)	rep71 (obj7)		(stateless) service object
process node 8	rep61 (obj6)	rep70 (obj7)		(stateless) service object

fault rarity (r):	21600 sec									
object replicas:	repX0 objX	repX1 objX	rep10 obj1	rep11 obj1	rep20 obj2	rep21 obj2	rep30 obj3	rep31 obj3	rep40 obj4	rep41 obj4
fault process (exp.)	2*r	2*r	2*r	2*r	r	r	r	r	2*r	2*r
restart times (exp.)	23.0	23.0	23.0	23.0	23.0	23.0	23.0	23.0	23.0	23.0
fault monitoring interval - periodic (sec)	15.0	15.0	15.0	15.0	15.0	15.0	15.0	15.0	15.0	15.0

Table 2. Optimal checkpoint placements and fault-affected response times (effectiveness)

request arrival rates (class-1 and class-2)		fault tolerance scheme	checkpoint interval obj1	checkpoint interval obj2	checkpoint interval obj3	checkpoint interval obj4	selected optimal placement	class-1 fault-affected	class-2 fault-affected	indicative QoS improvement
2.2	2.2	LDSC-based	**100** requests	**100** requests	**100** requests	**100** requests	**100 100 100 100**	115.2 sec	115.737 sec	88 %
2.2	2.2	PSC-based	**97** sec	**97** sec	**97** sec	**97** sec	**97 97 97 97**	129.291 sec	131.020 sec	83 %
2.4	2.4	LDSC-based	non significant	non significant	**12** requests	non significant	70 26 **12** 70	35.2019 sec	34.4863 sec	39 %
2.4	2.4	PSC-based	**97** sec	**23** sec	**23** sec	**97** sec	**97 23 23 97**	34.1743 sec	33.8495 sec	33 %
2.6	2.6	LDSC-based	non significant	**12** requests	**12** requests	non significant	56 **12 12** 56	26.2596 sec	25.6086 sec	46 %
2.6	2.6	PSC-based	non significant	**23** sec	**23** sec	non significant	60 **23 23** 60	24.7089 sec	24.2021 sec	41 %

For the four checkpoint placements (factors) and for three levels (namely 12, 56 and 100 for the LDSC-based scheme and 23, 60 and 97 for the PSC-based one) per factor, the chosen uniform design required only nine (9) runs in each load case. Table 2 summarizes the obtained optimal checkpoint placements, the selected optimal placement (the one with the minimum fault-unaffected response times) and the resulted fault-affected response times (effectiveness). PSC-based fault tolerance is equally effective to the LDSC-based one, in moderate load levels (arrival rates: 2.4 and 2.6), but falls short in high load levels (arrival rate: 2.2). The last column quantifies an indicative response time improvement, compared to the worst-case adjustments from those included in the experimental designs. Appropriate checkpoint placement results in response time reductions of up to 88%, for the heavy loaded system cases.

Finally, the tightest effective intervals based evaluation reveals the load levels (arrival rate: 2.2) for which the effectiveness of the tested schemes can be unacceptable. This behavior is related to the resulted queue lengths in the available service objects. Large queue lengths cause late request dispatching and in case of an object fault, the entire population of the queued requests contributes to the obtained mean. If a load level with the forenamed behavior cannot be excluded, this suggests the use of active replication in one or more objects. Table 3 summarizes the fault tolerance costs for the selected optimal checkpoint placements.

Table 3. Fault tolerance costs for the optimal checkpoint placements

request arrival rates (class-1 and class-2)		response times without f. t. (class-1)	response times without f. t. (class-2)	% overhead for the fault-unaffected requests (LDSC-based, class 1)	% overhead for the fault-unaffected requests (LDSC-based, class 2)	% overhead for the fault-unaffected requests (PSC-based, class 1)	% overhead for the fault-unaffected requests (PSC-based, class 1)
2.2	2.2	19.0543 sec	18,1990 sec	14.5 %	15.3 %	28.7 %	29.8 %
2.4	2.4	9.7565 sec	8.9463 sec	25.7 %	27.7 %	25.9 %	27.6 %
2.6	2.6	7.1186 sec	6.3596 sec	22.5 %	24.5 %	18.8 %	19.6 %

5 CONCLUSION

For (combined) fault tolerance (and load balancing) schemes, simulation metamodeling on the basis of small size uniform experimental designs allowed us to identify the significant checkpoint intervals and the optimal checkpoint placements with the minimum possible cost. The identified tightest effective intervals constitute the unique criterion that makes feasible the comparison of schemes composed of possibly different replication policies, checkpoint placement mechanisms and/or load balancing strategies.

Future research prospects include the development of a UML reliability-modeling framework like the one of [1] and the device of a coherent multiobjective optimization method, for fault tolerance performance evaluation.

REFERENCES

[1] Balsamo, S. & Marzolla, M. (2003). Simulation modeling of UML software architectures, *Proc. of the European Simulation Multiconference*, Society for Computer Simulation, Nottingham, 562-567

[2] Chen, X. & Lyu, R. (2003). Performance and effectiveness analysis of checkpointing in mobile environments, *Proc. of the 22nd IEEE Symposium on Reliable Distributed Systems (SRDS 03)*, Florence, Italy, 131-140

[3] Fang, K. T. & Lin, D. K. J. (2003). Uniform experimental designs and their applications in industry, *Mathematics Department Technical Report* No. 296, Hong Kong Baptist University

[4] Felber, P., Guerraoui, R. & Schiper, A. (2000). Replication of CORBA Objects, *Distributed Systems, Lecture Notes in Computer Science 1752*, Springer Verlag, 254-276

[5] Katsaros, P. & Lazos, C. (2003). A simulation test-bed for the design of dependable e-services, *WSEAS Transactions on Computers*, WSEAS Press, 4/2, 915-919

[6] Katsaros, P. & Lazos, C. (2004). Optimal object state transfer – recovery policies for fault tolerant distributed systems, *Proc. of the International Conference on Dependable Systems and Networks (DSN 04)*, IEEE Computer Society - IFIP, Florence, Italy

[7] Katsaros, P., Angelis, L. & Lazos, C. (2001). Applied multiresponse metamodeling for queuing network simulation experiments: problems and perspectives, *Proc. of the 4th EUROSIM Congress on Modelling and Simulation*, EUROSIM, Delfts, The Netherlands

[8] Litoiu, M., Rolia, J. & Serazzi, G. (2000). Designing process replication and activation: a quantitative approach, *IEEE Transactions on Software Engineering*, 26/12, 1168-1178

[9] Narasimhan, P., Moser, L. E. & Melliar-Smith, P. M. (2002). Strong replica consistency for fault-tolerant CORBA applications, *Journal of Computer Systems Science and Engineering*

[10] OMG (2001). Fault tolerant CORBA, *Object Management Group*, TC Doc. 2001-09-29, September 2001

[11] Szentiványi, D. & Nadjm-Tehrani, S. (2002). Building and evaluating a fault-tolerant CORBA infrastructure, *Proc. of the Workshop on Dependable Middleware-Based Systems (WDMS'02), International Conference on Dependable Systems and Networks (DSN 2002)*, Washington, DC, USA

[12] Uniform Design web pages (2000). http://www.math.hkbu.edu.hk/UniformDesign/