Performance and effectiveness trade-off for checkpointing in fault tolerant distributed systems

Panagiotis Katsaros Lefteris Angelis Constantine Lazos

Department of Informatics

Aristotle University of Thessaloniki

54124 Thessaloniki, Greece

tel.: +30-2310-998532, fax: +30-2310-998419

{katsaros, lef, clazos}@csd.auth.gr

Abstract

Checkpointing has a crucial impact on systems' performance and fault tolerance effectiveness: excessive checkpointing results in performance degradation, while deficient checkpointing incurs expensive recovery. In distributed systems with independent checkpoint activities there is no easy way to determine checkpoint frequencies optimizing response time and fault tolerance costs at the same time. The purpose of this paper is to investigate the potentialities of a statistical decision-making procedure. We adopt a simulation-based approach for obtaining performance metrics that are afterwards used for determining a trade off between checkpoint interval reductions and efficiency in performance. Statistical methodology including experimental design, regression analysis and optimization provides us the framework for comparing configurations, which use possibly different fault-tolerance mechanisms (replication-based or message logging based). Systematic research also allows us to take into account additional design factors, such as load balancing. The method is described in terms of a standardized object replication model (OMG FT-CORBA), but it is also possible to be applied in other (e.g. process based) computational models.

KEYWORDS: checkpointing and recovery, fault tolerance, distributed systems, performance evaluation, statistical analysis

1. Introduction

Checkpointing and message logging are used for minimizing loss of computation in the presence of non-recurrent faults. A checkpointing activity used in a software replication mechanism or in other fault tolerance mechanisms, saves the process state from time to time on stable storage. However, the main problem is that excessive checkpointing results

in performance degradation due to the computations involved, while deficient checkpointing incurs expensive recovery.

We focus on distributed systems, which use possibly different fault-tolerance mechanisms (replication-based or message logging based) for the constituent processes, with at least one and possibly more than one independent checkpointing activities. To the best of our knowledge, there are no analytic models for designing a checkpoint policy for the simultaneous optimization of the response time and the incurred fault tolerance cost. For this reason, aiming to study the complexity and the uncertainty in such systems, we investigate here the possibility of using a stochastic methodology including simulation, statistical analysis and modeling of the simulation results and finally optimization ([20]).

We introduce two separate performance metrics that reflect the system's performance and fault tolerance effectiveness. In order to identify the important factors that affect the performance metrics and to build the appropriate models based on them, the first phase of the proposed methodology involves the execution of simulation runs under various combinations of the factor levels. The factors related to checkpoint intervals are arranged in a systematic manner, using the appropriate experimental designs. As in practice there is no prior knowledge of the relationship between the factors and the performance metrics, we suggest the use of an appropriate class of experimental designs, namely the *uniform experimental designs*. The factor combinations along with the measurements of the considered performance metrics, which are collected from simulation runs, comprise the experimental data.

In the second stage, statistical analysis is applied to the experimental data. The main method used is regression analysis. Regression models are equations expressing a dependent variable in terms of some independent variables. In our case, the dependent

variable is the performance metric and the independent variables are the factors that affect it.

The last stage involves optimization of the regression models. This can be accomplished either by analytic or numerical methods and aims to reveal the shortest checkpoint intervals, below which no further gain is achieved in recovery time (Tightest Effective Checkpoint Intervals), taking into account other design factors like for example load balancing (load balancing is used as a mean to show the potentiality of the proposed approach to take into account additional design concerns, other than fault tolerance).

The whole procedure leads to a trade-off analysis that results either: (i) in the determination of checkpoint intervals fulfilling the response times goals at the lowest possible cost, or (ii) in checkpoint intervals showing the need for a different fault tolerance (and load balancing) scheme.

The method is described in terms of a standardized object replication context (OMG FT-CORBA), but the prototype simulator can be easily modified for use in other (e.g. process based) computational models. In distributed object systems ([28]), fault tolerance is attained via schemes that are possibly composed of different replication policies (active or passive replication with different checkpoint intervals) for the constituent objects.

For illustration purposes we used a synthetic workload scenario with a small number of experimental factors (4 checkpoint intervals), to make feasible the exploration of all possible combinations of two checkpointing mechanisms (load-dependent checkpointing and periodic checkpointing) and four load balancing strategies, in three different load levels. In total, we studied 24 system configurations based on a uniform design with 9 experimental cases for each of them (216 simulation experiments). The performed simulation output analysis was based on the method of independent replications in order to obtain 95% confidence intervals with width less than 3.5% of the estimated value.

From the results, we realized that fault tolerance performance is influenced by the complex method call dependencies between the system's replicated objects. Checkpointing affects the dispatching of the computation requests and influences the performance of the applied load balancing strategy.

These complicated nonlinear relationships can be successfully depicted by the small-size uniform experimental designs and validated by the prototype simulator, in an affordable CPU-time cost that depends on the number of experimental factors. The general methodology can also be applied to systems where the number of factors is large. In that case we can first employ a valid screening technique ([4]) in order to include in the analysis only those factors that seem to be the most important. The derived regression models constitute a mean for prediction, optimization, system administration and tuning based on varied performance requirements and system load levels.

Section 2 refers to related analytic and simulation-based performance evaluation techniques. Section 3 summarizes the computational model adopted in the developed prototype simulator, the simulated fault models and load balancing strategies, as well as, the assumed replication styles and fault detection mechanism. Section 4 provides a formal definition of the introduced performance metrics and the notion of the *tightest effective checkpoint intervals* (TECIs). Section 5 describes the proposed regression and optimization approach for estimating the TECIs. Section 6 presents the considered synthetic workload scenario and the obtained fault tolerance performance and effectiveness results. Section 7 introduces the proposed trade-off analysis that exploits the introduced performance metrics. The paper concludes with a discussion, on the results of the current work and the applicability of the proposed approach in large-scale systems.

2. Related work

In related bibliography, checkpointing is used in different fault tolerance protocols (coordinated or independent checkpointing, message logging, replication-based protocols) and also in process based or object based computational models.

Analytic performance models are mainly focused on single server checkpointing activities. Some representative and significant contributions are in [12], [26] and [35]. A more recent work that is reported in [11] also refers to checkpointing and recovering application state on a single compute resource (like in [14] and [15]), whereas our work focuses on problems of checkpointing and recovering jobs running simultaneously across multiple connected resources. In most analytic models, the primary aim is to predict the system availability ([13]) for a given checkpoint interval, while our work deals with the problem of determining a *set of checkpoint intervals* taking into account response time and performance optimization requirements.

In [5], the authors introduce an analytic model for the optimal checkpoint interval that minimizes the application execution time and maximizes the average fault tolerance effectiveness, in a mobile environment. The overall approach is focused on the Telecom Wireless CORBA mobile architecture [29].

In [23] the authors propose a hybrid mathematical programming and analytic evaluation algorithm for a different trade-off problem: to determine process replication or threading levels, in order to avoid unnecessary queuing delays for clients and unnecessary high consumption of memory.

A closely related simulation-based approach has been recently published in [30]. The performance metric integrated in the presented simulation tool is used for relative performance comparison of different checkpointing and recovery protocols. A protocol's performance is assessed from

(a) the overhead due to checkpointing and recovery activities and

(b) the quality of recovery, measured by the amount of computation lost due to rollback in case of a fault.

The performance metric introduced in our work does not directly account the quality of recovery (as in [30]), but the effects of recovery in computations' average response time.

We emphasize that we do not aim to introduce an integrated simulation tool like in [30] and [31]. We simply present here a prototype simulator that was used to obtain experimental data for the proposed statistical approach.

This work's most important contribution is the notion of TECIs and a novel combination of well-known and advanced statistical techniques (regression analysis and optimization based on uniform experimental designs) to determine them. The TECIs is the single criterion that makes feasible the comparison between different fault tolerance schemes in terms of their effectiveness. TECIs are formally defined based on the introduced performance metrics, which allow trading the gains of a checkpoint interval reduction, against the incurred overhead. Preliminary results of the present work were given in [19].

3. The prototype simulator

3.1 Computational model

The computational model adopted in our prototype simulator is the OMG Core Object Model - as it is documented in [27] - enriched by the assumptions specified in the OMG FT-CORBA standard ([28]). We outline the assumptions that we think are necessary for the presentation of the used synthetic workload scenario and the applied decision-making approach:

- An object can model any kind of entity or concept and is characterized by its distinct object identity, which is immutable, persists for as long as the object

exists and is independent of the object's properties or behavior. Each object owns or does not own a state and is accessed via a set of methods.

- Each method has a signature consisting of its name, the set of parameters and the set of results. A method describes an action that can be applied to parameters. A *method invocation*, also called *request*, can list some parameters on behalf of a requester (client) and can cause the method to return results. The methods of an object constitute the only way to change its state.
- We only consider the use of *synchronous method invocations*: the object requiring the execution of the invoked method (requester) stops executing and waits for the invoked execution to terminate and the reply to return. Upon reception of the *reply*, the sender resumes. A computation invoked in a server application object (*service object*), will be called *service request* and is possible to involve one or more synchronous and possibly nested *requests* to other objects.
- We adopt the *"at-most-once" invocation semantics* of the OMG Core Object Model, which means that each request is executed at most once. Duplicate method invocations due to replication are detected and suppressed.
- In OMG FT-CORBA, fault-tolerance is based 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.
- We do not place any assumptions about the network topology. For the protocols making up the interprocess communication we adopt the OMG FT-

CORBA assumption that they provide *reliable*, *totally ordered delivery of requests* to the replicas of each object.

- *Strong replica consistency* (OMG FT-CORBA): even in the presence of faults, as members of an object group execute a sequence of methods invoked on the object group, its behavior is logically equivalent to that of a single fault-free object processing the same sequence of method invocations. For each object, strong replica consistency retains an appropriate context that depends on the object group's *replication style* (*active*, *warm passive* or *cold passive*).
- We exclude the possibility of multithreaded Object Request Brokers ([33]), in which case the order of the dispatched methods would have to be rendered deterministic ([1], [2]). However, the proposed decision-making approach is also *open to take into account multithreading* (if present).
- We assume deterministic behavior of the underlying operating system. Ordering of dispatched method invocations at the application programming level is not allowed. Finally, the application either does not make use of system calls returning processor-specific information or such calls are handled by an appropriate mechanism without introducing non-determinism.
- *Load balancing* has been used as a mean to investigate the potentiality of the proposed approach to take into account additional design concerns, other than fault tolerance. We do not distinguish between centralized or distributed load balancing. Load information (if applicable) is provided by the underlying middleware infrastructure, which also works out request assignment to the available service objects in a transparent and fault tolerant manner (as in [22] and [34]).

3.2 Simulated fault models

We model faults that do not recur after recovery and are eventually manifested as transient object faults (OMG FT-CORBA). Application objects conform to the *fail-stop model* ([32]), which means that they fail by crashing, without emission of spurious messages. Commission faults as for example Byzantine faults, where an object generates incorrect results, are not addressed.

As in the OMG FT-CORBA specification, we exclude network-partitioning faults that separate the hosts of the system into two or more sets.

In general, when a service object fails, the already queued requests are not lost. All requests arriving while the object is down are queued. For the passively replicated objects that own a state we have also implemented a number of different *omission fault scenarios* (loss behavior) and a set of request-retry policies.

The prototype tool features an extensible object-oriented design that allows:

- to take into account different fault propagation scenarios
- the use of alternative object fault - repair distributions
- the application of load-dependent fault models, like those used in [16].

3.3 Simulated load balancing strategies

We have implemented a set of load balancing strategies with no need of state synchronization, in order to preserve consistency across participating service object groups. The decision of *request assignment* to a specific service object may be based on different modes: per-request or per-session.

In the used workload scenario we considered only the case of *per-request load balancing* in passively replicated and stateless service objects, where there is no need for checkpointing. The four load balancing strategies that we tested are described below:

- *PROB*: random probabilistic with equal probabilities for the service objects,

- *TB*: threshold based, where request assignment to another service object is activated when the number of queued requests exceeds the specified threshold parameter (Figure 1a),
- *RR*: assignment on a round robin basis
- *EQL*: assignment to the service object with the smallest number of queued requests (Figure 1b)

3.4 Replication styles

We are free to consider different replication styles ([3], [10], [17], [25], [28]) for the constituent objects, with at least one checkpointing activity (passively replicated object).

In active replication, all of the object group replicas execute each invocation independently, but in the same order. The individual replicas maintain exactly the same state and in case of a fault in one member, the simulated application continues with the results provided by the others, without having to wait for fault detection and recovery.

Cold or warm passive replication assumes that during fault-free operation, only one member - the *primary* (Figure 2a) - of the object group executes the methods invoked on the group. The state of the primary (for the objects that own state) and the sequence of the invoked methods are recorded in a *message log*, according to the specified checkpoint interval parameter.

Strong replica consistency implies that, at the end of each checkpoint (transitions P:ST→P:N and B:ST→B:N in Figure 2), all members of the object group have the same state or have access to the same state. Upon detection of an occurred fault (state P:F in Figure 2a), a *backup replica* is promoted to be the new primary (transition B:N→P:R). The state of the new primary is restored to the state of the old one by reloading the last saved checkpoint and subsequently reapplying the request messages that have been recorded in the log. This implies that a client can re-invoke a request on a server and receive a reply to

that request, without the risk that the method will be executed more than once ("at-most-once" invocation semantics).

In cold passive replication, the backup replicas are not activated. When the current primary fails, a new one is selected and then activated. In warm passive replication, the backups have been already activated and their states are continuously synchronized with the primary replica's state, according to the specified frequency of *state transfers*.

A checkpoint/state transfer may be postponed when the primary is in-between an invocation service or it happens to be blocked, waiting for a response. In the course of a checkpoint or a state transfer activity, *new invocations may be received, but they cannot be processed, before the end of it*.

The prototype simulator allows modeling of the objects interaction effects regarding:

- the simultaneous resource possession, caused by the synchronous and often nested object invocations,
- the hardware resource contention, as a result of the replicas distribution,
- the extra load and the queued requests' blocking costs caused by the recurrent checkpointing/state transfer activities,
- the extra load caused by a replica restart and replay of the logged requests,
- the overhead assumed for re-invocation of requests that are possibly lost because of a recipient omission fault and
- the overhead assumed for periodically checking, if an object is faulty or not (section 3.5).

3.5 Fault detection

The modeled fault detection setting assumes the existence of a fault tolerant fault monitoring service. In general, the fault detector that monitors an application object is

usually located, for efficiency, on the same host as that of the object. A global fault detector that is replicated for fault tolerance monitors the local fault detectors.

It has been found ([11]) that fault monitoring causes an approximate 5% increment, in the processor (of a Pentium-II based 200+ MHz machine) utilization, for about 500 milliseconds. In our simulator, this overhead is taken into account. Each object is *periodically checked*, according to a specified time interval that represents the sum of the fault monitoring interval plus the time allowed for subsequent response from the object, to determine whether it is faulty.

4. Performance, effectiveness and tightest effective checkpoint intervals

System's performance depends on the overhead incurred due to checkpointing and recovery activities. Fault tolerance effectiveness refers to the quality of recovery, i.e. the amount of computation lost due to rollback in case of a fault. We designed two separate metrics that reflect the system's performance and fault tolerance effectiveness and have been proved suitable for the applied statistical analysis. In this section we give precise definitions of them in terms of the described computational model. However, the employed formal description (Table 1 provides a glossary of the used notation) and the metrics definitions can be easily adapted to other computational models.

In our setup, a distributed system is composed of multiple objects o_1, o_2, . . ., o_n. Each object o_i, $1 \leq i \leq n$ is replicated according to

- an active replication policy with k object replicas, $o_i^{ar_1}, o_i^{ar_2}, ..., o_i^{ar_k}$ or alternatively,
- the passive replication policy of Figure 2, with one primary replica o_i^{prim} and one backup o_i^{back}.

An *object's methods* $op_l^{o_i}$, $1 \le l \le$ #(methods of o_i) may be synchronously invoked by remote method invocations. A method invocation is represented by an ordered pair of a request and a reply message ($rq(op_l^{o_i})$, $rp(op_l^{o_i})$), for simplicity denoted ($rq_l^{o_i}$, $rp_l^{o_i}$).

On receipt of a *request* $rq_l^{o_i}$ the request is queued in the queues of all replicas o_i^r, denoted by $Q(o_i^r)$, where $r \in \{ar_j \mid 1 \le j \le \#(o_i^r)\} \cup \{prim\}$. Each o_i^r is placed on a separate object server and queued requests are served in FIFO ordering under a single thread of control. However, the proposed approach can also be extended for application in other multithreading cases. If *r= prim*, executed requests $rq_l^{o_i}$ are appended to a *local log queue* log_i.

Let $op_{l,p}^{o_i}$ ($p \in \mathbf{N}$) denote a method invocation instance of $op_l^{o_i}$. Each $op_{l,p}^{o_i}$ may further invoke another method instance $op_{s,p}^{o_t}$, with $1 \le s \le$ #(methods of o_t), $1 \le t \le n$, $t \ne i$ and this is the case of a *nested invocation*, where $op_{l,p}^{o_i}$ is blocked up to the reception of $rp_{s,p}^{o_t}$. Access to a simulated object's state is performed by primitive read/write messages that induce computational resource consumption, but they are not included in the methods' message sequence specifications.

A *message sequence specification* for $op_l^{o_i}$, $1 \le l \le$ #(methods of o_i) is given as a total order relation < over the set

$$MsgSeq(op_l^{o_i}) = \{ op_s^{o_t} \mid 1 \le s \le \#(\text{methods of } o_t),\ 1 \le t \le n \text{ and } t \ne i \}$$

of nested invocations generated by $op_l^{o_i}$. This set is empty when $op_l^{o_i}$ causes exclusively read/write operations on the state of the simulated o_i and no nested invocations. A method $op_{s1}^{o_{t1}}$ is referred to as *preceding another method* $op_{s2}^{o_{t2}}$ ($op_{s1}^{o_{t1}} < op_{s2}^{o_{t2}}$) if and only if $rq_{s2}^{o_{t2}}$ may be sent only after $rp_{s1}^{o_{t1}}$ is already received.

A *global state S* of the system consists of all *local log queues* log_i and local queues

$$Q(o_i^r) \text{ with } r \in \{ar_m \mid 1 \le m \le \#(o_i^r)\} \cup \{prim, back\} \text{ and } 1 \le i \le n$$

plus the values of $3\sum_{i=1}^{n} \#(o_i^r)$ additional boolean variables that are defined as follows:

- $crash_i^r$: these variables are initially false and become true at the time that o_i^r becomes faulty.
- $failed_i^r$: these variables are initially false and become true when the system detects that $crash_i^r$ = true.
- $recovered_i^r$: these variables are initially true to indicate that o_i^{prim} reflects the object state of having executed all $rq_{s,q}^{o_i} \in log_i$, with $1 \leq s \leq$ #(methods of o_i). In any other case $recovered_i^r$ is false.

An initial global state S_0 is the global state in which all $Q(o_i^r)$ and log_i do not have queued requests (empty) and the additional boolean variables are set to their initial values. An *event* is an action that changes the global state of the system from S to S'. We will use the notation $e(S) = S'$ to denote that e occurs in global state S and results in global state S'.

We specify *chk_init*, *send*, *respond*, *crash*, *failed*, *restart* and *chk_end* event types by the following notation:

- *chk_init*(o_i^{r1}, o_i^{r2}) denotes the event whereby a *checkpoint/state transfer request* $c_i^{r1,r2}$ for o_i^{r1} and o_i^{r2} is placed second in the order of queues $Q(o_i^{r1})$ and $Q(o_i^{r2})$ (or at the head of them in case of empty queue(s)).
- *send*$(o_i^{r1}, o_j, rq_{l,p}^{o_j})$ denotes the event whereby o_i^{r1} sends $rq_{l,p}^{o_j}$ to all queues

$$Q(o_j^{r2}),\ r_2 \in \{ar_m \mid 1 \leq m \leq \#(o_j^r)\} \cup \{prim\},$$

$p \in \mathbf{N}$ and $op_l^{o_j}$ is a member of the message sequence of the request at the head of $Q(o_i^{r1})$. A send event changes the local queue(s) $Q(o_j^{r2})$ if $rq_{l,p}^{o_j} \notin Q(o_j^{r2})$ and $crash_j^{r2}$ is false, by appending the request message $rq_{l,p}^{o_j}$. If $rq_{l,p}^{o_j} \notin Q(o_j^{r2})$ and $crash_j^{r2}$ is true then $Q(o_j^{r2})$ changes or does not change depending on the applied omission fault scenario (section 3.2). Finally, the event *timeout*$(o_i^{r1}, rq_{l,p}^{o_j})$ is scheduled to occur, if it is required by the applied request-retry policy.

- *timeout*($o_i^{r1}, rq_{l,p}^{o_j}$) denotes the event whereby a request-retry timeout for $rq_{l,p}^{o_j}$ occurs. The event *send*($o_i^{r1}, o_j, rq_{l,p}^{o_j}$) takes place and a new *timeout*($o_i^{r1}, rq_{l,p}^{o_j}$) is then scheduled.
- *resp*($o_i^{r1}, o_j^{r2}, rp_{l,p}^{o_j}$) denotes the event whereby o_j^{r2} responds with $rp_{l,p}^{o_j}$ to o_i^{r1} for some $r_2 \in \{ar_m \mid 1 \le m \le \#(o_j^r)\} \cup \{prim\}$. Scheduled *timeout*($o_i^r, rq_{l,p}^{o_j}$) events (if any) are canceled for all $r \in \{ar_m \mid 1 \le m \le \#(o_i^r)\} \cup \{prim\}$ and not only for r_1. The local queue $Q(o_j^{r2})$ changes by removing $rq_{l,p}^{o_j}$ from the head and if $r_2 = prim$ then $rq_{l,p}^{o_j}$ is appended to log_j. If the next request is some $rq_{s,q}^{o_j}$, then o_j^{r2} proceeds to executing $rq_{s,q}^{o_j}$, based on *MsgSeq*($op_s^{o_j}$). If the next request is some $c_j^{r3,r2}$, then o_j^{r2} is blocked until $c_j^{r3,r2}$ is also placed at the head of $Q(o_j^{r3})$. A *chk_end*($c_j^{r3,r2}$) event is then scheduled to occur.
- *crash*(o_i^r) denotes the event whereby $crash_i^r$ becomes true. This models the occurrence of a fault in o_i^r. Potential *resp*($o_i^r, o_j^{r1}, rp_{l,p}^{o_j}$) and *timeout*($o_i^r, rq_{l,p}^{o_j}$) events are ignored. If $r = prim$ then $Q(o_i^r)$ is changed according to the applied omission fault scenario.
- *failed*(o_i^r) denotes the event whereby $failed_i^r$ becomes true. A recovery of o_i^r is then set up: in all replication cases, a *restart*(o_i^r) event is scheduled.

  In the passive replication of Figure 2, if $r = prim$ and $crash_i^{back}$ is false, then $Q(o_i^r)$ is copied to $Q(o_i^{back})$, o_i^{back} becomes o_i^{prim} and r becomes *back*. This change results in

  $$failed_i^{prim} = crash_i^{prim} = \text{false and } failed_i^{back} = crash_i^{back} = \text{true}$$

  If $\#(rq_{s,q}^{o_i} \in log_i \mid 1 \le s \le \#(\text{methods of } o_i)) > 0$ then $recovered_i^{prim}$ becomes false and the variable keeps this value while o_i^{prim} has not yet replayed all $rq_{s,q}^{o_i} \in log_i$. When $recovered_i^{prim}$ changes to true, log_i is emptied.
- *restart*(o_i^r) denotes the event whereby $crash_i^r$ and $failed_i^r$ become false.

- *chk_end*($c_i^{r1,r2}$) denotes the event whereby $c_i^{r1,r2}$ is removed from o_i^{r1} and o_i^{r2}. If r_1, r_2 are not *back*, they proceed to the execution of the next request $rq_{s,q}^{o_j}$ based on *MsgSeq*($op_s^{o_j}$).

All forenamed events are atomic and each event affects only the relevant local queues and the relevant boolean variables. Thus, if $crash_i^r$ is false in the global state *S*, when *send*($o_i^r, o_j, rq_{l,p}^{o_j}$) occurs, then $crash_i^r$ is also false in the resulting global state of *S´*.

Definition 4.1: A *run of the system* is an infinite sequence of global states

$$run = (S_0, S_1, S_2, \ldots)$$

where S_0 is an initial global state and there exists a sequence of events (e_0, e_1, e_2, . . .) such that $\forall i \geq 0$, $e_i(S_i) = S_{i+1}$.

The *history of run* is the sequence of events $H_{run} = (e_0, e_1, e_2, \ldots)$ such that $\forall i \geq 0$, $e_i(S_i) = S_{i+1}$.

For any run *run*, H_{run} is uniquely determined and also, *run* can be constructed from the history H_{run} and the initial global state S_0.

4.1 Fault tolerance performance and effectiveness

We do not directly account the amount of computation lost due to rollback in a recovering object (quality of recovery), but instead, the effects of recovery in computations' average response time. This is achieved by explicitly distinguishing the service requests in two classes:

- the *fault-unaffected requests*, which include the service requests that are not affected by the occurred faults and
- the *fault-affected requests*, which include the service requests that are affected by the occurred faults.

Fault-unaffected requests comprise the vast majority of the dispatched service requests and their average response time is an adequate measure of the observed fault tolerance cost (performance). The fault-affected service requests reflect the effectiveness of the applied fault tolerance scheme regarding the quality of recovery. The proposed decision-making approach trades the gains of a potential improvement in the second class of service requests, against the overhead imposed to the first class of service requests.

Definition 4.2: A synchronous request $rq_{l,p}^{o_j}$ to a passively replicated object *is affected by a fault at* o_j^{prim} in *run* and the boolean predicate $F\_AFFECTED(rq_{l,p}^{o_j}, o_j)$ becomes true if and only if

$$H_{run}=(e_0; x; e_u= send(o_i^r, o_j, rq_{l,p}^{o_j}); y; e_v= resp(o_i^r, o_j^{prim}, rp_{l,p}^{o_j}); w)$$

where x, y finite sequences of events with $e_u \notin x$, e_v is the first *resp* event and w an infinite sequence of events such that either:

i. $S_u \models \neg crash_j^{prim}$ and $\#(crash(o_j^{prim}) \in y) > 0$ or

ii. $S_u \models (crash_j^{prim} \vee \neg recovered_j^{prim})$ or

iii. $S_u \models (\exists\, rq_{s,q}^{o_j} \in Q(o_j^{prim}), 1 \le s \le \#(\text{methods of } o_j)$ such that $S_v \models F\_AFFECTED(rq_{s,q}^{o_j}, o_j))$

If $S_v \models F\_AFFECTED(rq_{l,p}^{o_j}, o_j)$ and $S_v \models (\exists\, rq_{d,p}^{o_i}$ at the head of $Q(o_i^r)$, such that $op_l^{o_j} \in MsgSeq(op_d^{o_i})$, with $i \le n$, $i \ne j$, $r \in \{ar_m \mid 1 \le m \le \#(o_i^r)\} \cup \{prim\}$, $1 \le d \le \#(\text{methods of } o_i)$), then $S_{v+1} \models F\_AFFECTED(rq_{d,p}^{o_i}, o_j)$.

The first condition reflects the case whereby $rq_{l,p}^{o_j}$ is sent to an operational o_j^{prim} and $crash_j^{prim}$ changes to true before the occurrence of the expected *resp* event. The second condition reflects the case whereby $rq_{l,p}^{o_j}$ is sent to a o_j^{prim}, that is not yet operational as a result of a $crash(o_j^{prim})$ event. The third condition reflects the case whereby $rq_{l,p}^{o_j}$ is queued

behind a $rq_{s,q}^{o_j}$ that is eventually affected by a *crash*(o_j^{prim}) event. If $rq_{l,p}^{o_j}$ is a nested invocation generated by $rq_{d,p}^{o_i}$ at o_i^r, then $rq_{d,p}^{o_i}$ is also affected by the *crash*(o_j^{prim}) event.

Definition 4.3: A synchronous request $rq_{l,p}^{o_j}$ to an actively replicated object *is affected by a fault at some* o_j^{r1} in *run* and the boolean predicate *F_AFFECTED*($rq_{l,p}^{o_j}$, o_j) becomes true if and only if

$$H_{run}=(e_0; x; e_u= send(o_i^r, o_j, rq_{l,p}^{o_j}); y; e_v= resp(o_i^r, o_j^{r2}, rp_{l,p}^{o_j}); w)$$

where *x*, *y* finite sequences of events with $e_u \notin x$, e_v is the first *resp* event with $r_2 \in \{ar_m \mid 1 \le m \le \#(o_j^r)\}$ and *w* is an infinite sequence of events such that either:

i. $\#(chk\_end(c_j^{r1,r2}) \in y) > 0$ and $r_1 \neq r_2$ or

ii. $\#(\text{crash}(o_j^{r1}) \in y) > 0$ with $r_2 = r_1$ or

iii. $S_u \models (\exists\, rq_{s,q}^{o_j}$ in all $Q(o_j^{r1})$, $r_1 \in \{ar_m \mid 1 \le m \le \#(o_j^r)\}$, $1 \le s \le \#$(methods of o_j), such that $S_v \models F\_AFFECTED(rq_{s,q}^{o_j}, o_j)$).

If $S_v \models F\_AFFECTED(rq_{l,p}^{o_j}, o_j)$ and $S_v \models (\exists\, rq_{d,p}^{o_i}$ at the head of $Q(o_i^r)$, such that $op_l^{o_j} \in MsgSeq(op_d^{o_i})$, with $i \le n$, $i \neq j$, $r \in \{ar_m \mid 1 \le m \le \#(o_i^r)\} \cup \{prim\}$, $1 \le d \le \#$(methods of o_i)), then $S_{v+1} \models F\_AFFECTED(rq_{d,p}^{o_i}, o_j)$.

The first condition reflects the case whereby $rq_{l,p}^{o_j}$ has been blocked, in order to realize a state transfer for a replica recovery. The second condition reflects the case whereby o_j^{r1} is the first replica involved in a $resp(o_i^r, o_j^{r1}, rp_{l,p}^{o_j})$ event, but has previously become faulty in *y*. The third condition reflects the case whereby $rq_{l,p}^{o_j}$ is queued behind another request $rq_{s,q}^{o_j}$ that is already affected by a fault at some o_j^{r1}.

Definition 4.4: A synchronous request $rq_{l,p}^{o_j}$ *is included in the class of the fault-affected requests* in *run* if and only if

$$H_{run}=(e_0; x; e_u= send(o_i^r, o_j, rq_{l,p}^{o_j}); y; e_v= resp(o_i^r, o_j^{r2}, rp_{l,p}^{o_j}); w)$$

where x, y finite sequences of events with $e_u \notin x$, e_v is the first *resp* event with $r_2 \in \{ar_m \mid 1 \le m \le \#(o_j^r)\} \cup \{prim\}$ and w is an infinite sequence of events such that either:

i. $S_v \models F\_AFFECTED(rq_{l,p}^{o_j}, o_j)$ or

ii. $S_v \models (\exists\ op_s^{o_i} \in MsgSeq(op_l^{o_j}),\ 1 \le s \le \#(\text{methods of } o_i),\ 1 \le i \le n,\ i \ne j$ and $F\_AFFECTED(rq_{s,p}^{o_i}, o_t)$ for some $1 \le t \le n,\ t \ne j)$

iii. $\exists\ resp(o_t^{r1}, o_j^{r2}, rp_{s,q}^{o_j}) \in y$ for some o_t^{r1}, $1 \le s \le \#(\text{methods of } o_j)$ such that

$$S_v \models F\_AFFECTED(rq_{s,q}^{o_j}, o_u), \text{ for some } 1 \le u \le n$$

Events $resp(o_i^r, o_j^{r3}, rp_{l,q}^{o_j}) \in w$ with $r_3 \ne r_2$ do not change the classification of $rq_{l,q}^{o_j}$.

The first condition reflects the case whereby $rq_{l,p}^{o_j}$ is affected by a fault at some o_j^r. The second condition reflects the case whereby one of the nested invocations generated by $rq_{l,p}^{o_j}$ is affected by a fault at o_t, for some $1 \le t \le n$, $t \ne j$. Finally, the third condition reflects the case whereby $rq_{l,p}^{o_j}$ is queued behind a $rq_{s,q}^{o_j}$ that is eventually affected by a fault at o_u, for some $1 \le u \le n$. All other $rq_{l,p}^{o_j}$ ($p \in \mathbf{N}$) in *run*, *are included in the class of the fault-unaffected requests*.

4.2 The tightest effective checkpoint intervals (TECIs)

Definition 4.5: The *checkpoint interval* I_i for a passively replicated object o_i determines the times of subsequent *chk_init*(o_i^{prim}, o_i^{back}) events, in all runs of the system, according to the selected *checkpointing mechanism*:

i. In a *load dependent checkpointing mechanism* (LDSC), I_i is specified as an integer that if

$H_{run}=(e_0; x; e_u=chk\_init(o_i^{prim}, o_i^{back}); y; e_v=chk\_init(o_i^{prim}, o_i^{back}); w)$

is the history of *run* such that $\neg(\exists\ chk\_init(o_i^{prim}, o_i^{back}) \in y)$ then

$S_v \models \#(rq_{l,p}^{o_i} \in log_i, 1 \le l \le \#(\text{methods of } o_i), p \in \mathbf{N}) = I_i$

ii. In a *periodic checkpointing mechanism* (PSC), I_i is specified as a positive real that if

$H_{run}=(e_0; x; e_u=chk\_init(o_i^{prim}, o_i^{back}); y; e_v=chk\_init(o_i^{prim}, o_i^{back}); w)$

is the history of *run* such that $\neg(\exists\ chk\_init(o_i^{prim}, o_i^{back}) \in y)$ then

$$T(e_v) = T(e_u) + I_i,$$

with *T(e)* denoting the time of occurrence of *e*.

Definition 4.6: In a fault tolerance scheme that includes z passively replicated objects, $1 \le z \le n$, a *checkpoint interval reduction* from I_i to $I_i - rd_i$ (for some o_i, $1 \le i \le n$) is *effective*, if resulting in a reduction of the average response time for the fault-affected service requests. If in a set of z checkpoint intervals there is no I_i amenable to an effective interval reduction, this set specifies the *tightest effective checkpoint intervals* (TECIs) for the considered fault tolerance scheme.

The TECIs correspond to the system configuration with the minimum average response time for the fault-affected service requests (fault tolerance effectiveness). Their

estimation is an applied optimization problem with a number of factors that coincides with the number of independent checkpointing activities. In systems with a large number of checkpointing activities, we can first employ a valid screening technique ([4]), in order to include in the analysis only those factors that are found to be the most important.

The *TECIs is the single criterion that makes feasible the comparison between different fault tolerance schemes* in terms of their effectiveness. It is also possible to take into account other interdependent system design concerns like for example load balancing. TECIs can be used:

- to find out the average response time of the fault-unaffected service requests (fault tolerance performance), for the considered fault tolerance and load balancing scheme combination in its *optimum effectiveness configuration*,
- to assess and possibly compare the potentiality of the studied scheme(s) for being adapted in varied performance and effectiveness requirements and varied system load levels,
- to support the decision-making approach of section 7, in order to determine checkpoint intervals that guarantee a response time goal and at the same time minimize the incurred fault tolerance cost.

We found out that *the TECIs depend on the system load*. Also, in the performed experiments and comparisons we discovered prominent performance and effectiveness differences in high system loads (close to the system thrashing level).

Different object replica distributions to the available nodes, as well as object fault models with different fault - repair distributions and dependencies raise the need for new TECIs estimation.

5. Statistical analysis and finding of TECIs

The search for the TECIs involves three stages:

- the experimental design,
- the experiment's execution and the finding of a suitable model based on the obtained results and
- the study of the model found, for discovering the checkpoint interval parameters that minimize the average response time of the fault-affected service requests.

5.1 The experimental design

In this stage, the primary goal is to explore the nature of the effect that checkpoint interval parameters have on the fault-affected service response times. We employ an experimental design to arrange the prefixed levels of the considered factors, so as to ensure that the estimations obtained by the analysis of the experimental results - *in a relatively small number of runs* - are as accurate as possible.

There is a vast literature concerning the right choice of an experimental design and there are many types of designs available in tables (factorial, fractional, etc), so that the analyst can easily choose a design "off the shelf" according to his/her needs ([6]). Most of the experimental designs assume that there is some prior knowledge of the underlying model and that the purpose of the experiment is to estimate the model parameters (usually the regression coefficients) with the highest possible accuracy. In our problem however, this is hardly true and the analyst needs to conduct several successive experiments in order to get an idea of the underlying model.

Recently, a new class of experimental designs has been proposed for industrial and computer experiments, where the underlying model is either unknown or too complicated. These are the so-called uniform experimental designs [9]. They are space-filling designs

with experimental points scattered uniformly on the domain. They are used to 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 large number of them are already available in tables ([36]). In our problem, an appropriately selected uniform design allows identifying the unknown effects of the experiment's factors, on the average of the fault-affected service response times.

Suppose that there are z factors namely $I_1,\ldots, I_z$, for which we want to explore their effect on the response variable Y. Suppose also that their relation can be described by the regression model

$$Y = g(I_1,\ldots, I_z) + \varepsilon$$

where g is an unknown function and ε is the random error. In order to estimate the function g we need to collect data for various factor values combinations to get the corresponding output values of Y. We represent each factor by a number q of predefined values (called *levels*), which is the same for all factors. Then, the total number of level combinations and therefore the number of experiments is q^z, which in general is a prohibitively large number.

Traditionally, when the modeled relation is known to be linear, fractional factorial and orthogonal designs are used. When, as in our case, there is no prior information regarding the function g, we prefer the use of a uniform design that is usually denoted by $U_n(q^z)$, where n is the number of experiments to be decided by the analyst. Typically, a $U_n(q^z)$ design is represented by a $n \times z$ array with elements from $\{1, \ldots, q\}$ denoting the levels of the considered factors. The levels are arranged in an appropriate way in order to satisfy certain uniformity criteria. For details regarding the construction and the implementation of uniform experimental designs we refer to [9].

5.2 The building of a model

After the execution of the designed experiment, the obtained data are further analyzed in order to build a model. The model will describe and explain the relation between the average response time for the fault-affected service requests (dependent variable *Y*) and the checkpoint intervals (independent variables or factors $I_1,\ldots, I_z$) that are supposed to affect the dependent variable. As we have already mentioned in section 5.1, the regression model we are trying to estimate is

$$Y = g(I_1,\ldots, I_z) + \varepsilon$$

Usually we make some assumptions regarding function *g()*, i.e. we assume that the relation is or at least can be transformed to a linear expression of the form

$$y = b_0 + \sum_{i=1}^{k} b_i x_i + \varepsilon$$

where the unknown parameters b_i can be estimated from the data. This is known as a *parametric model*. Note that he new variables y and x_i can be transformations, products or powers of the original dependent variable and the factors $I_1,\ldots, I_z$ (of course, there are regression models that cannot be transformed to a linear form). In case that function *g()* is assumed to be completely unknown, the model is called *nonparametric* and this function can be estimated from the data by various methods (see [18] for details). Although nonparametric regression is suitable for a variety of cases, the form of the resulting equation is not convenient for further analysis and for this reason we used parametric regression, as the final equation can be more easily studied for optimization.

In an ideal situation, the whole model-building procedure would be fully automated. Unfortunately, due to the complexity of the relations and the interactions between the factors, one has to try various variable transformations and tests, in order to conclude to a single representative model. This model has to be both explanatory and predictive and for this reason it is necessary to calculate measures and to perform diagnostic tests, in order to assess the model's validity. For example, the fitting of an equation to data is measured by

the coefficient of determination R^2, but it is always useful to examine the distribution of the residuals (difference between observed and the estimated from the model values) for potential violation of the assumptions, outlying values etc.

The statistical methodology for fitting equations to data is well-known as regression analysis ([8]). We usually start with simple linear models and quite often an adequate model comes up via transformations of the original variables. Common transformations are the logarithmic, the power and the root transformations of the dependent or the independent variables. In certain cases where interactions between the independent variables are apparent, there may be a need for operations between them, such as multiplication or division.

Another usual problem in the procedure of building a model is the determination of the most important independent variables for the model and the elimination of the variables that are not important for the model. In order to exclude those variables that do not have effect on the dependent variable and those which are highly correlated with others and therefore their effect can be explained by variables already in the model, we can use a range of algorithms available by statistical programs, applying *stepwise regression*. These are sequential procedures that by adding or removing variables can eventually identify the most significant factors and build a model that fits efficiently on the obtained experimental data.

Apart from the fitting accuracy, an important issue is the interpretation of the model. The model has to be intuitively sound regarding the reasonableness of the equation and its outcomes. It is therefore essential to verify that the relations described by the model are realistic and also, to compare the results of the simulated system to that of the equation, especially in domains of local minima or maxima, which are important for the optimization.

5.3 Model optimization: the TECIs

Having concluded to an adequate model that fits the data, the next step is to search for the TECIs, i.e. the combination of the factor levels that minimizes the average for the fault-affected response times.

In general, there are many optimization techniques for different problems and their choice depends on the nature of the problem, i.e. the function properties and the constraints set on the factors values (the domain of the function). In most cases we can use calculus-based methods such as Lagrange multipliers or steepest descent algorithms, but for more complicated problems with a large number of factors (checkpoint intervals), there are the heuristic random search algorithms such as simulated annealing and evolutionary algorithms.

As the optimization takes place on a model fitted on experimental data, there is always the possibility of having unreasonable results with respect to the optimization of the simulated system. A common recommended practice is to always verify the results of the statistical analysis, by conducting further experiments, in the area around the found TECIs. The combination of regression analysis and optimization provides us insight and understanding of the simulated system.

6. A synthetic workload scenario and its TECIs

We utilize a synthetic workload scenario with a small number of experimental factors (checkpoint intervals), to make feasible the exploration of all possible combinations of two checkpointing mechanisms and four load balancing strategies, in three load levels. In total, we performed 216 simulation runs to study 24 system configurations.

The system model (Figure 3) used includes four (4) interacting objects that own state (`obj1`, `obj2`, `obj3`, `obj4`) and four (4) stateless service objects (instances of the

class `SrvRequestAccepting`). Invoked service requests belong to one of the two considered classes (methods) with their own message sequence specifications.

We assume passive object replication for all objects, with one primary and one backup per object group. There is no need for checkpointing in the stateless service objects. The objects that own state are involved in regular checkpointing/state transfer activities as specified by the state machines of Figure 2. State transfers result in backup state updates (in case of a live backup replica) and in persistent checkpoints and take place in a frequency specified by the considered checkpoint interval parameter. Thus, the performance and the effectiveness of the applied fault tolerance scheme depend on a quadruple, where each number specifies the used checkpoint interval parameter for the corresponding state owning object.

Table 2 specifies the used system model parameters and the assumed object replicas allocation. Request assignment to the four service objects (`obj0`, `obj5`, `obj6`, `obj7`) is performed on the basis of the applied load balancing strategy (PROB, TB(5), RR and EQL). The studied system load levels are specified by the service request arrival distributions, for the two classes of service requests. We also assume different CPU service times (different object methods) for the two classes of requests and an associated overhead, when executing log-replayed re-invoked requests. CPU time consumption for the checkpointing/state transfer activities depends on the object state size and on the state transfer time per KB, as they are specified in the table: a state transfer between two object replicas (rep`OBJ`0 and rep`OBJ`1, where `OBJ` corresponds to an object id) is performed in the speed of the slowest participating object replica. CPU time consumption for replicas placed in the same host is performed on a processor-sharing basis.

Table 3 specifies the considered parametric object fault model. The assumed fault rarity parameter (r) can be changed in order to study the sensitivity of the TECIs in varied

fault rarity conditions. Object restart times are exponentially distributed and are activated on the detection of an object fault at the end of a fault-monitoring interval. We did not consider load dependent fault rates and fault propagation to the collocated object replicas.

The tested load balancing and fault tolerance scheme combinations are distinguished with respect to the applied checkpointing mechanism:

- the first four schemes employ load-dependent checkpoint intervals (LDSC), combined respectively with the PROB, the TB (with parameter 5), the RR and the EQL load balancing strategy and
- the other four schemes employ periodic checkpoint intervals (PSC) and are also combined with the forenamed load balancing strategies.

For the four checkpoint interval parameters (factors) and the levels

- 12, 56 and 100 for the LDSC-based schemes and
- 23, 60 and 97 for the PSC-based schemes,

the selected uniform experimental design required only nine (9) simulation runs, for each case of load balancing and system load level.

The subsequent stepwise regressions yielded effectiveness prediction models with sufficiently high coefficients of determination (R^2 of over 90%). We report the fitted regression model for the EQL load balancing case with LDSC-based checkpoint intervals and service request interarrival times with rates 2.2:

$$Y = -61.373 + \frac{234949.5}{I_1 * I_2} + \frac{6917.75}{I_3} + 0.044 * I_1 * I_3 - 0.016 * I_1 * I_2 ,$$

Y denotes the sum of the average fault-affected response times for both classes of invoked service requests and I_1, I_2, I_3 the checkpoint interval parameters found to have a statistically significant impact. The calculated coefficient of determination (R^2) is 0.997.

The described analysis detected the statistically significant factors and the subsequent minimization of Y yielded the corresponding TECIs. For service request

interarrival times with rates 2.2 (high system load), table 4 summarizes the optimization results and the average fault-affected response times in the found optimum effectiveness configurations. For the non-significant checkpoint intervals (the significant ones are shown in bold), we selected the factor level minimizing the average fault-unaffected response time.

We observe differences,

- in the obtained optimum effectiveness configurations (`obj3` checkpoint interval parameter), with respect to the used fault tolerance and load balancing scheme combination and
- in the resulted average fault-affected response times (fault tolerance effectiveness).

The results for the two other system load cases revealed a dependence of the TECIs on the system load level. On the other hand, we did not observe notable differences in the average fault-affected response times (effectiveness), as in the high system load case (Figure 4).

Fault tolerance schemes may exhibit different effectiveness behavior in higher system load levels and are thus characterized by different potentialities for adaptation in varied system loads. If one increases the frequencies of service request arrivals, throughput will increase up to a point, then drop. The user observes this as a sudden increase in response time. This phenomenon is similar to thrashing in operating systems.

Figure 4 revealed the load level (service request interarrival times with rates 2.2), where the fault-affected response times exhibit a sudden increase. Thus, the TECIs *are also used to assess and possibly compare* the potentiality of the studied scheme(s) for being adapted in varied system loads.

Figure 5 summarizes comparative performance results with no fault tolerance and with the fault tolerance provided by the found TECIs. In the second case, fault tolerance performance is quantified by the average fault-unaffected response time. The O(1) RR load balancing strategy performed almost as well as the EQL strategy that is O(# service objects) and does not scale as RR, because of the accompanied load-info gathering costs.

7. Fault tolerance trade-off analysis

This section introduces a trade-off analysis in order to determine the checkpoint intervals satisfying a response time goal with the minimum possible cost. The analysis trades the gains (in the fault-affected response times) of checkpoint interval reductions, against the overhead incurred to the vast majority of the dispatched computations, which are not affected by the occurred faults. If the procedure ends without having satisfied the set response time goal, the used fault tolerance and load-balancing scheme combination is not appropriate.

To assess a checkpoint interval reduction we take into account three possibilities:

- plain gain, i.e. improved average response times for both the fault-affected and the fault-unaffected computations,
- improved average response time for the fault-affected computations, at the cost of a measurable worsening of the fault-unaffected ones or
- plain worsening of both average response times.

The trade-off analysis is applied only to the checkpoint intervals found to have significant impact to the average response time for the fault-affected computations. This makes the proposed procedure appropriate for fault tolerance schemes with a large number of checkpointing activities, where the significant ones might be for example only four or five. We assume that checkpoint intervals are initially set to values that are close to the

factor level minimizing the average fault-unaffected response time. The checkpoint intervals that are not found to have significant impact to the average fault-affected response time keep the values used in the initial fault tolerance setting.

The analysis is a step-by-step procedure, where the interval reduction performed in the initial scheme is decided, based on the potential gains or improvements, for all interval reduction possibilities, when considering two test cases per significant factor. We do not test reduction values, which result in checkpoint intervals smaller than those included in the TECIs.

Plain gains are quantified by the sum of the differences between the corresponding means for the fault-affected and the fault-unaffected response times. Worsening cases are not taken into account. In all other cases, quantification is performed on the basis of the criterion ([21]):

$$trade-off\ ratio = \frac{avg_{flc} - avg_{fmc}}{avg_{omc} - avg_{olc}}$$

where:

avg_{flc} = the average response time of the fault-affected computations for the initial scheme

avg_{fmc} = the average response time of the fault-affected computations, for the scheme with the reduced checkpoint interval

avg_{omc} = the average response time of the fault-unaffected computations, for the scheme with the reduced checkpoint interval

avg_{olc} = the average response time of the fault-unaffected compuations, for the initial scheme

If there are interval reductions that result in plain gains, we choose the one with the maximum gain (sum of the differences between the corresponding means). In any other case, we choose the interval reduction with the maximum trade-off ratio, if it is not less

than a specified threshold. If the maximum trade-off ratio is less than the specified threshold, the corresponding interval reduction results in negligible effectiveness improvement with unacceptably high overhead. In that case, the procedure ends with the checkpoint intervals of the last step.

The interval reduction values that are tested in each step are adjusted as follows: we use the same values for the checkpoint intervals that are not affected by the last interval reduction or if the last step is a plain gain reduction. In any other case, we use the value of the last performed reduction and its half value.

The procedure converges to the checkpoint intervals satisfying the response time goal for the fault-affected computations, with the minimum possible cost. If not possible, this becomes evident since we end with the TECIs or with trade-off ratios less than the specified threshold. In that case, the used fault tolerance and load-balancing scheme combination is not appropriate.

Table 5 summarizes the results of the described trade-off analysis for the LDSC-based fault tolerance scheme of Figure 2, when combined with the RR load-balancing strategy. The service request interarrival times are exponential with rates 2.4 for both classes of them. The response time goal for the fault-affected service requests is an average of less than 33 sec.

Only the checkpoint intervals I_2 and I_3 were found to have significant impact to the considered performance metric (when the request interarrival times are exponential with rates 2.2 all checkpoint intervals I_1 to I_4 matter). In the initial fault tolerance scheme all checkpoint intervals are set to 90 requests, a value that is close to the factor level minimizing the average fault-unaffected response time. The interval reductions (rd_i) tested for each significant checkpoint interval are 20 and 40 requests, two values that do not outflank the found TECIs. We ignore trade-off ratios less than 2.5 (threshold parameter).

In step 1, we observe no improvements when reducing I_2. On the other hand, the maximum trade-off ratio is given, when $rd_3 = 40$ and the checkpoint intervals tested in next step are given by the quadruple 90-90-50-90. The interval reductions tested in step 2 are the same values for I_2 and the value of the last performed reduction (40) and its half value for I_3. The maximum trade-off ratio is given, when reducing I_2 with $rd_2 = 20$. The obtained quadruple 90-70-30-90 does not satisfy the fault-affected response time goal (33 sec). Thus, the procedure ends with one more step, where the maximum trade-off ratio is given for $rd_2 = 10$. The checkpoint intervals 90-60-30-90 represent a fault tolerance scheme with an effectiveness improvement of about 30%, which satisfies the set response time goal and an increase of only 4.1% in the average of the fault-unaffected service requests.

8. Conclusion

We presented a statistical decision-making approach for distributed systems with independent checkpointing activities. The computational model used in the description is a standardized object replication framework. Design issues other than fault tolerance, as for example load balancing, may be taken into account.

We defined two separate response time metrics, which independently quantify fault tolerance performance and effectiveness. We employed a formal description framework that can be also adapted to other computational models. Since there is no prior knowledge of the relationship between the experiment factors and the metrics, we suggest the use of a uniform experimental design to collect the experimental data. The derived regression models can be used as a mean for prediction, optimization, system administration and tuning based on varied performance requirements.

We introduced the notion of the tightest effective checkpoint intervals (TECIs) and we described how they are estimated. The TECIs is the single criterion that makes feasible the comparison between different fault tolerance schemes in terms of their effectiveness.

In our experimentation with the considered synthetic workload scenario, TECIs allowed the comparison of diverse fault tolerance (with load-dependent or periodic checkpoint intervals) and load balancing scheme combinations. The obtained results revealed a dependence of the TECIs on the system load level. In high load levels we observed notable differences in the effectiveness behavior of the considered fault tolerance and load balancing scheme combinations. Thus, TECIs were also used as a mean to assess the potentiality of the studied schemes for being adapted in varied system load levels. Regarding the observed fault tolerance performance, we found that the RR load balancing strategy performed almost as well as the EQL strategy that does not scale as RR, because of the accompanied load-info gathering costs.

Finally, we introduced a trade-off analysis that is based on the defined response time metrics and the found TECIs. The proposed analysis trades the gains (in fault tolerance effectiveness) of checkpoint interval reductions, against the incurred overhead. It converges to the checkpoint intervals that satisfy a response time goal with the minimum possible cost. For the considered fault tolerance and load-balancing scheme we succeed an effectiveness improvement of about 30% accompanied with an increase of only 4.1% in the average of the fault-unaffected computations. The analysis is also applicable in fault tolerance schemes with a large number of checkpointing activities, since potential interval reductions are tested only for the few statistically significant checkpointing activities.

The presented performance and fault tolerance effectiveness evaluation can potentially be the cornerstone of UML-based performance models ([24]) of dependable component-based systems ([7]). The main purpose of the proposed statistical approach is to

provide a framework for comprehensive analysis of fault tolerance schemes, with independent checkpoint activities. The analysis framework is generic and apart from a simulation-based dataset is also possible to use a measurement-based dataset. As a future research work, we consider the prospect to collect actual traces for different system load scenarios and then feed them to the simulation and apply the proposed analysis, in order to assess its validity and the resulted performance gains.

Acknowledgments

We acknowledge the anonymous referees for their helpful comments, which contributed to improving the quality of the article.

References

[1] Basile C., Whisnant K., Kalbarczyk Z., Iyer R. Loose synchronization of multithreaded replicas. *Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS 02)*, Suita, Japan, 2002; 250-255.

[2] Basile C., Kalbarczyk, Z., Iyer R. A preemptive deterministic scheduling algorithm for multithreaded replicas. *Proceedings of the IEEE/IFIP Int. Conference on Dependable Systems and Networks (DSN 03)*, San Francisco, 2003; 149-158.

[3] Bessani A. N., Fraga, J. S., Lung, L. C., Alchieri, E. A. P., Active Replication in CORBA: Standards, Protocols, and Implementation Framework. *Proceedings of the International Symposium on Distributed Objects and Applications (DOA 2004)*, Lecture Notes in Computer Science 3291, 2004; 1395 - 1412

[4] Bettonvil B., Kleijnen J. P. C. 1997. Searching for important factors in simulation models with many factors: Sequential bifurcation, *European Journal of Operational Research*, **96** (1): 180-194.

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

[6] Colbourn J. C., Dinitz J. H. *The CRC handbook of combinatorial designs*. CRC Press Inc., 1996.

[7] Crnkovic I., Larsson, M., Preiss, O., Concerning predictability in dependable component-based systems: classification of quality attributes, Lecture Notes in Computer Science 3549, Springer, 2005; 257-278.

[8] Draper N., Smith H. *Applied regression analysis*. 2nd edition, Wiley, 1981.

[9] 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.

[10] Felber P., Guerraoui R., Schiper A. Replication of CORBA Objects. *Distributed Systems (Lecture Notes in Computer Science, vol. 1752).* Springer: Berlin, 2000; 254-276.

[11] Garg S., Huang Y., Kintala C. M. R., Trivedi K. S., Yajnik S. Performance and reliability evaluation of passive replication schemes in application level fault tolerance. *Proceedings of the 29th IEEE Int. Symposium on Fault-Tolerant Computing*, Madison, Wisconsin, USA, 1999; 322-329.

[12] Gelenbe, E. 1979. On the optimum checkpoint interval. *Journal of the Association for Computing Machinery*; 26 (2): 259-270

[13] Gelenbe, E., Finkel, D., Tripathi, S. K. On the availability of a distributed computer system with failing components. *Proceedings of the 1985 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, Austin, Texas, USA, 1985; 6-13

[14] Global Grid Forum. *Use-cases for grid checkpoint and recovery*. GridCPR-WG, November 2004.

[15] Global Grid Forum. *An architecture for grid checkpoint and recovery (GridCPR) services and a GridCPR Application Programming Interface*. GridCPR-WG, September 2005.

[16] Goswami K. K., Iyer R. K., Young L. 1997. DEPEND: A simulation-based environment for system level dependability analysis. *IEEE Transactions on Computers*; **46** (1): 60-74.

[17] GroupPac project, http://grouppac.sourceforge.net/grouppac/en/index.html, 2005

[18] Hardle W. *Applied nonparametric regression.* Cambridge University Press, 1993.

[19] Katsaros P., Lazos C. Optimal object state transfer – recovery policies for fault tolerant distributed systems. *Proceedings of the IEEE/IFIP Int. Conference on Dependable Systems and Networks (DSN 04)*, IEEE Computer Society Press: Los Alamitos, CA, 2004; 762-771.

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

[21] Krishna C. M., Shin K. G., Yann-Hang Lee. 1984. Optimization criteria for checkpoint placement. *Communications of the ACM*; **27** (10): 1008-1012.

[22] Lindermeier M. Load management for distributed object-oriented environments. *Proceedings of the International Symposium on Distributed Objects and Applications (DOA'00)*, IEEE Computer Society, 2000.

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

[24] Marzolla, M, Simulation-based performance modeling of UML software architectures, Dottorato di Ricerca in Informatica, II Ciclo Nuova Serie, Dipartimento di Informatica, Università Ca' Foscari di Venezia, 2003

[25] 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*, **17** (2): 103-114.

[26] Nicola, V. F., Spanje, J. M. 1990. Comparative analysis of different models of checkpointing and recovery. *IEEE Transactions on Software Engineering*, 16 (8): 807-821.

[27] Object Management Group. *Object Management Architecture Guide*. revision 3.0, OMG Technical Committee Document ab/97-05-05, June 1995.

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

[29] Object Management Group. *Telecom wireless CORBA*. OMG Technical Committee Document dtc/2001-06-02, June 2001.

[30] Paul H. S., Gupta A., Badrinath R. 2003. Performance comparison of checkpoint and recovery protocols. *Concurrency and Computation: Practice and Experience*, **15**: 1363-1386.

[31] Ramamurthy B., Upadhyaya S. J., Iyer R. K. An object-oriented test-bed for the evaluation of check-pointing and recovery systems. *Proceedings of the 27th IEEE Int. Symposium on Fault-Tolerant Computing*, Seattle, USA, 1997; 194-203.

[32] Schlichting R. D., Schneider F. B. 1983. Fail-Stop processors: An approach to designing fault-tolerant computing systems. *ACM Transactions on Computer Systems*, **1** (3): 222-238.

[33] Schmidt D. C. 1998. Evaluating architectures for multithreaded object request brokers. *Communications of the ACM*, **41** (10): 54-60.

[34] Schnekenburger T., 2000. Load balancing in CORBA: A survey of concepts, patterns and techniques. *The Journal of Supercomputing*, **15**: 141-161, Kluwer Academic.

[35] Tantawi, A. N., Ruschitzka, M., 1984. Performance analysis of checkpointing strategies. ACM Transactions on Computer Systems, 2 (2): 123-144.

[36] Uniform Design web pages, http://www.math.hkbu.edu.hk/UniformDesign/, 2000

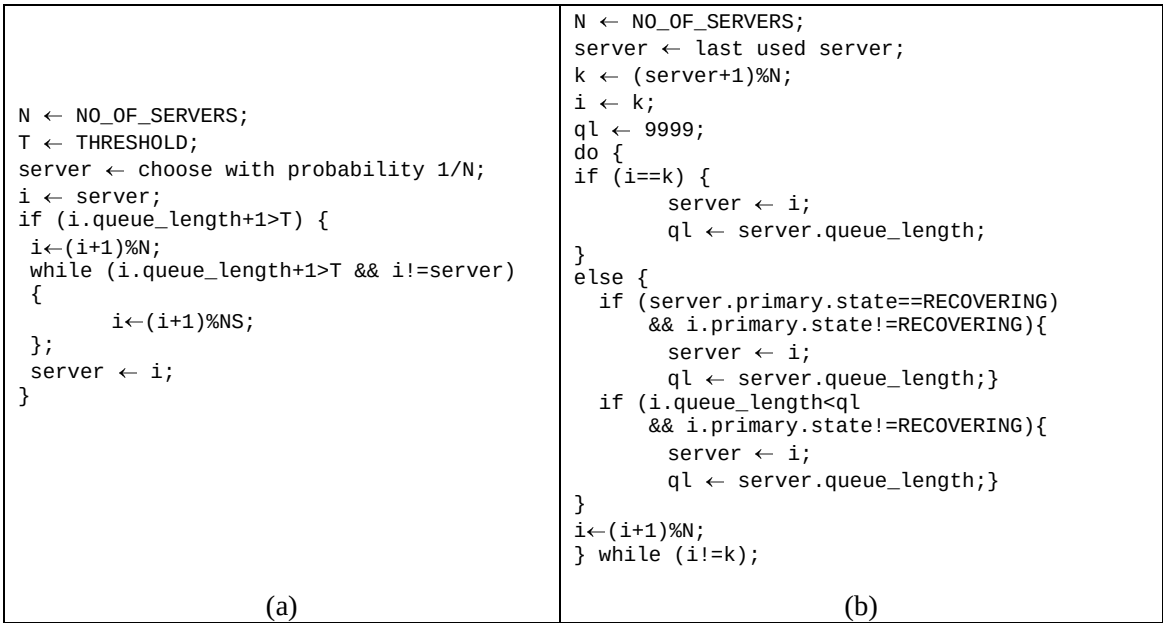
Biographies

Panagiotis Katsaros is a Lecturer of Computer Science in the Department of Informatics, Aristotle University of Thessaloniki (A.U.Th.), Greece. He holds a Diploma in Mathematics and a PhD in Computer Science from A.U.Th. and a Master of Science in Software Engineering from University of Aston in Birmingham, UK. He has published research articles in international journals and conference proceedings in the areas of fault tolerance, software performance and security modeling, simulation output analysis, compilers design and software engineering.

Lefteris Angelis received his B.Sc. and Ph.D. degree in Mathematics from Aristotle University of Thessaloniki (A.U.Th.). He works currently as an Assistant Professor at the Department of Informatics of A.U.Th.. His research interests involve statistical methods with applications in software engineering and information systems, computational methods in mathematics and statistics, planning of experiments and simulation techniques.

Constantine Lazos is emeritus professor of Computer Science of the Aristotle University of Thessaloniki, Greece. He holds a B.Sc. in Mathematics from the University of Athens, a M.Sc. from the University of Birmingham and a Ph.D from the University of Southampton, both in Computer Science. His main research interests lie in the area of computer system modeling and simulation, performance evaluation and simulation methodology. He was a lecturer of the University of Birmingham for two academic years and a visiting professor of the same University. He has published papers and is reviewer in international journals and in international conferences proceedings. He is the author of 10 books all in computer science.

Illustrations



```
N ← NO_OF_SERVERS;
T ← THRESHOLD;
server ← choose with probability 1/N;
i ← server;
if (i.queue_length+1>T) {
 i←(i+1)%N;
 while (i.queue_length+1>T && i!=server)
 {
        i←(i+1)%NS;
 };
 server ← i;
}
```

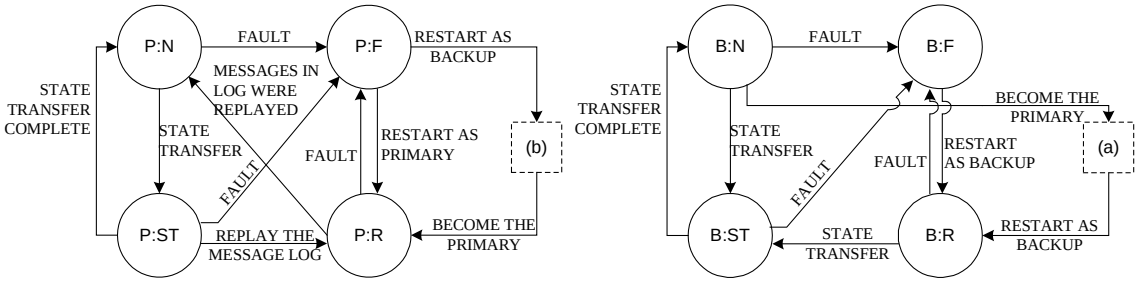
(a)

```
N ← NO_OF_SERVERS;
server ← last used server;
k ← (server+1)%N;
i ← k;
ql ← 9999;
do {
if (i==k) {
        server ← i;
        ql ← server.queue_length;
}
else {
  if (server.primary.state==RECOVERING)
      && i.primary.state!=RECOVERING){
        server ← i;
        ql ← server.queue_length;}
  if (i.queue_length<ql
      && i.primary.state!=RECOVERING){
        server ← i;
        ql ← server.queue_length;}
}
i←(i+1)%N;
} while (i!=k);
```

(b)

Figure 1. The TB (a) and EQL (b) load balancing strategies

Panagiotis Katsaros



P:N	Primary Normal
P:F	Primary Failed
P:R	Primary Recovering
P:ST	Primary in State Transfer

(a) the primary object replica

B:N	Backup Normal
B:F	Backup Failed
B:R	Backup Recovering
B:ST	Backup in State Transfer

(b) a backup object replica

Figure 2. Warm passive replication with one primary and one backup object

Panagiotis Katsaros

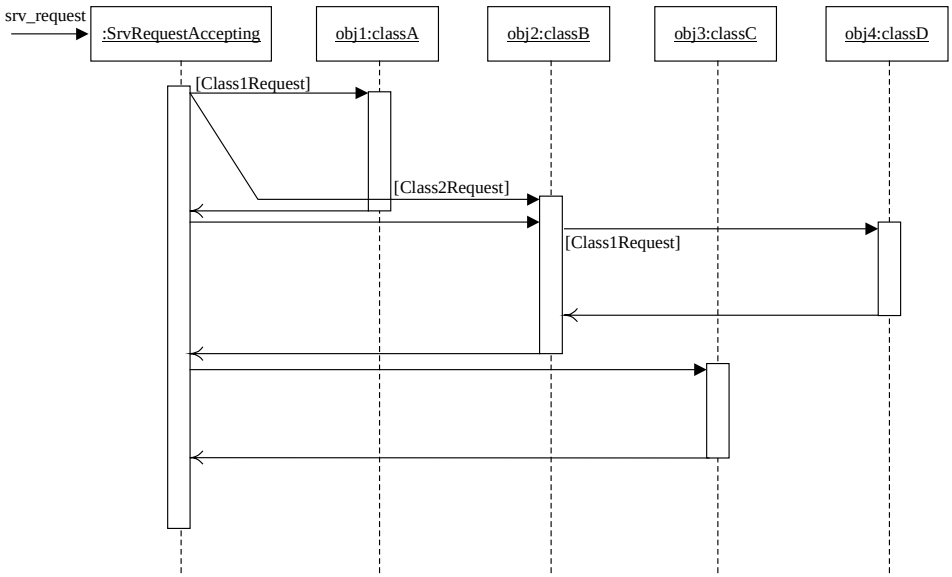


Figure 3. Message sequence for the objects of the case system

Panagiotis Katsaros

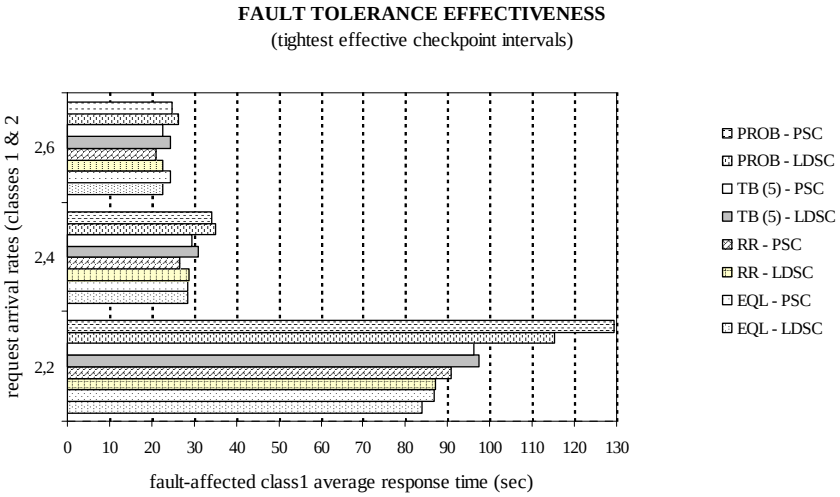
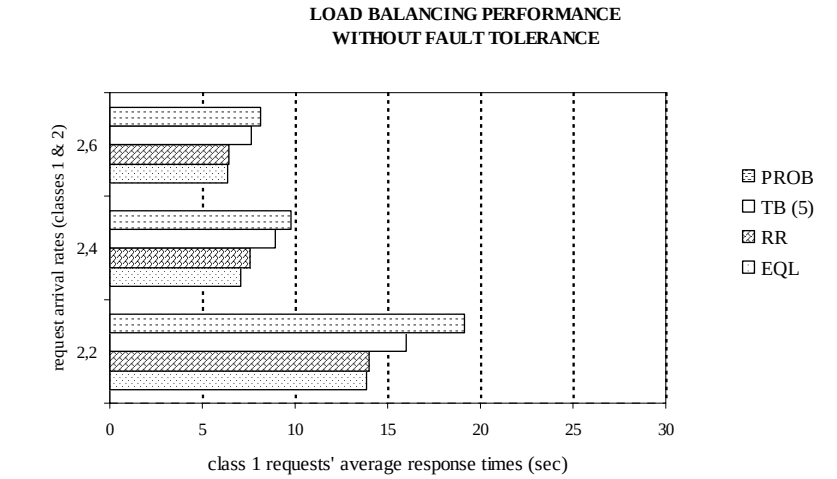


Figure 4. Fault tolerance effectiveness

Panagiotis Katsaros



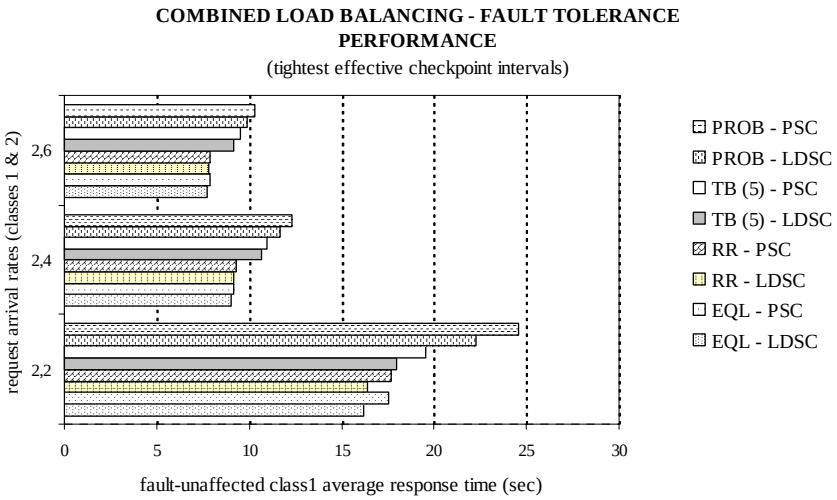


Figure 5. Fault tolerance performance

Panagiotis Katsaros

Tables

Table 1 Glossary of notation

o_i^r	object replica r of o_i	$crash_i^r$	there is an object fault at o_i^r
ar_j	active replica j of some object	$failed_i^r$	a fault at o_i^r has been detected
prim	the primary object replica of a passively replicated object	$recovered_i^r$	o_i^r reflects the object state of having executed all $rq_{s,q}^{o_i} \in log_i$
back	the backup object replica of a passively replicated object	$c_i^{r1,r2}$	request for checkpointing/state transfer between o_i^{r1} and o_i^{r2}
$op_{l,p}^{o_i}$	a method instance of $op_l^{o_i}$ with $1 \le l \le$ #(methods of o_i)	*run*	an infinite sequence of global states $S_0, S_1, \ldots$
$(rq_{l,p}^{o_i}, rp_{l,p}^{o_i})$	an ordered pair of request and reply messages that collectively represent the execution of $op_{l,p}^{o_i}$	H_{run}	history of *run*: a sequence of events $e_0, e_1, \ldots$ such that $\forall i \ge 0$, $e_i(S_i) = S_i + 1$
log_i	the log queue of a passively replicated o_i	$S_i \models \varphi$	predicate φ holds in global state S_i
$Q(o_i^r)$	the local queue of requests in o_i^r	$F\_AFFECTED(rq_{l,p}^{o_i}, o_j)$	
$MsgSeq(op_l^{o_i})$	the totally ordered set of nested invocations $op_s^{o_t}$ generated by $op_l^{o_i}$		boolean predicate indicating that $rq_{l,p}^{o_i}$ is affected by a fault at o_j

Panagiotis Katsaros

Table 2 System model parameters

service objects:	objX:SrvRequestAccepting (X=0, 5, 6, 7)			
load balancing:	PROB, TB(5), RR, EQL			
class 1 service request interarrival times:	exponential with rates	2.6	2.4	2.2
class 2 service request interarrival times:	exponential with rates	2.6	2.4	2.2

	obj1:classA	**obj2:classB**	**obj3:classC**	**obj4:classD**
object state sizes (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 requests service times (exponential with means)	0.52	0.57	0.6	0.6	0.83	0.83	0.32	0.32	0.2	0.2
class 2 requests service times (exponential with means)	-	-	0.28	0.28	0.83	0.83	-	-	0.2	0.2
log-replayed re-invoked requests (exponential with means)	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	-	-
state transfer times -sec/KB (exponential)	0.8	0.8	0.6	0.6	0.6	0.6	0.8	0.8	-	-
omission fault scenario	no request loss	no request loss	no request loss	no request loss	no request loss	no request loss	no request loss	no request loss	no request loss	no request loss

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

Table 3 Object fault model

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 interarrival times (exponential with means)	2*r	2*r	2*r	2*r	r	r	r	r	2*r	2*r
restart times (exponential with means)	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

Panagiotis Katsaros

Table 4 Tightest effective checkpoint intervals and fault-affected response times (effectiveness)

request interarrival times with rates (class-1 & class-2)		**fault tolerance scheme**	**load bala-ncing**	I_1	I_2	I_3	I_4	***TECIs***	**fault-affected response times: average for the class-1 requests**	**fault-affected response times: average for the class-2 requests**
2.2	2.2	LDSC-based	PROB	**100** requests	**100** requests	**100** requests	**100** requests	**100 100 100 100**	115.2 sec	115.7 sec
2.2	2.2	PSC-based	PROB	**97** sec	**97** sec	**97** sec	**97** sec	**97 97 97 97**	129.3 sec	131.0 sec
2.2	2.2	LDSC-based	TB(5)	**100** requests	**100** requests	**100** requests	**100** requests	**100 100 100 100**	97.5 sec	97.4 sec
2.2	2.2	PSC-based	TB(5)	**97** sec	**97** sec	**60** sec	non significant	**97 97 60** 97	96.2 sec	96.9 sec
2.2	2.2	LDSC-based	RR	**100** requests	**100** requests	**100** requests	**100** requests	**100 100 100 100**	87.0 sec	87.4 sec
2.2	2.2	PSC-based	RR	**97** sec	**97** sec	**60** sec	non significant	**97 97 60** 97	90.9 sec	91.1 sec
2.2	2.2	LDSC-based	EQL	**100** requests	**100** requests	**40** requests	non significant	**100 100 40** 100	83.9 sec	84.5 sec
2.2	2.2	PSC-based	EQL	**97** sec	**97** sec	**60** sec	non significant	**97 97 60** 97	86.8 sec	87.3 sec

Table 5 Trade-off analysis for the LDSC-based scheme, RR load balancing and service request interarrival times with rates 2.4 – goal: average fault-affected response time less than 33 sec

	class 1 fault-affected response time (average)	class 2 fault-affected response time (average)	average	class 1 fault-unaffected response time (average)	class 2 fault-unaffected response time (average)	average	trade-off ratio	rd_1	rd_2	rd_3	rd_4
initial checkpoint intervals ($I_1 - I_2 - I_3 - I_4$):											
90-90-90-90	46,37	46,16	46,27	7,94	7,08	7,51					
step 1 interval reductions:											
90-90-50-90	38,93	38,13	38,53	8,05	7,18	7,62	69,68	-	-	40	-
90-90-70-90	42,71	42,21	42,46	8,02	7,15	7,58	47,54	-	-	20	-
90-50-90-90	47,49	46,54	47,01	no improvement				-	40	-	-
90-70-90-90	47,32	46,57	46,95	no improvement				-	20	-	-
checkpoint intervals:											
90-90-50-90	38,93	38,13	38,53	8,05	7,18	7,62					
step 2 interval reductions:											
90-90-30-90	37,06	36,38	36,72	8,17	7,31	7,74	14,89	-	-	20	-
90-90-10-90	36,80	35,56	36,18	9,10	8,20	8,65	2,27	-	-	40	-
90-50-50-90	36,36	36,14	36,25	8,18	7,31	7,74	18,25	-	40	-	-
90-70-50-90	36,93	36,12	36,53	8,11	7,23	7,67	39,32	-	20	-	-
checkpoint intervals:											
90-70-50-90	36,93	36,12	36,53	8,11	7,23	7,67					
step 3 interval reductions:											
90-70-30-90	35,16	34,35	34,75	8,23	7,38	7,81	12,91	-	-	20	-
90-70-10-90	35,21	35,04	35,13	8,80	7,90	8,35	2,05	-	-	40	-
90-50-50-90	36,36	36,14	36,25	8,18	7,31	7,74	3,71	-	20	-	-
90-60-50-90	37,19	36,72	36,96	no improvement				-	10	-	-
checkpoint intervals:											
90-70-30-90	35,16	34,35	34,75	8,23	7,38	7,81					
step 4 interval reductions:											
90-70-10-90	35,21	35,04	35,13	no improvement				-	-	20	-
90-70-20-90	32,96	32,11	32,54	8,38	7,51	7,94	16,24	-	-	10	-
90-50-30-90	33,14	31,95	32,54	8,38	7,50	7,94	16,27	-	20	-	-
90-60-30-90	32,91	32,00	32,45	8,24	7,38	7,81	307,13	-	10	-	-