Configuring Fault-Tolerant Servers for Best Performance

Diana Szentiványi, Simin Nadjm-Tehrani
Department of Computer and Information Science, Linköping University, Sweden
{diasz,snt}@ida.liu.se

John M. Noble
Department of Mathematics, Linköping University, Sweden
jonob@mai.liu.se

Abstract

When using primary-backup replication, one checkpoints the primary's state to reduce the failover time to a backup, upon failure of the primary. A trade-off is involved: by frequent checkpointing the response time for requests during "no-failure" intervals is increased. On the other hand, when checkpointing frequency is low, failover takes longer. In this paper we provide a theoretical ground for computing the optimal checkpointing interval that minimizes the average response time of requests, given parameters such as load, failure rate and service rates. We use queuing theory for modelling the support for failover in a system where a logging server records client update operations. The novelty of the model is inclusion of the backlog processing time for requests accumulated during failover. Also, our model considers the waiting times in the queue of the logging server as well as the application server.*

1 Introduction

To ensure high availability of a server system that may crash, three fault tolerance mechanisms are commonly used: replication by primary-backup, checkpointing and a failure recovery mechanism. Although high availability is a core requirement, obtaining fast enough response to a client query is also a major concern of the architects of such systems.

Checkpointing is a mechanism for reading and logging the state of the primary. In a typical fault-tolerant server system, *update* [2] calls (client requests) arriving at the primary are also logged, in a so-called *call log* so that the state of a crashed server can be reconstructed by replaying the calls since checkpointing. After each checkpointing, the content of the call log is *pruned* to contain only the records of calls that arrived after the checkpoint request. When the primary crashes, *failover* takes place, i.e. the checkpointed state is transferred to the back-up, and the calls present in the log are replayed on the back-up server. The time taken to reconstruct the state of the crashed server is called *failover time*. There is a major trade-off between the checkpointing frequency, and the length of the failover time. If checkpointing is done often, then the chance to have pruned the log shortly before the failure, and thus to have low failover time, is increased. However, during no-failure periods, when the checkpointing request is served on the servers, client requests have to wait.

In this paper we use mathematical analysis employing queuing theory, to configure a fault-tolerant server system for best *average response time*. We determine an expression of the *checkpointing interval* that minimizes average response time, given load, failure rate and other parameters of the system, like e.g. service rates. The model is an extension of a basic model that was developed for maximizing a server's availability interval [11]. The idea is to support the system architect by providing the machinery so that the above parameters are input and the checkpointing interval that leads to minimal average response time is obtained. It further helps in making an informed rather than empirical choice of the checkpointing interval.

The main contribution of the paper is the detailed modelling including *backlog processing* time, i.e. the extra time spent to process the requests that queue up during failover. This time is crucial in the study of the minimal average response time. Another contribution of the work is to show how long the maximum average response time for the service can become if the load, in terms of client request arrival rate, is increased beyond the value for which the infrastructure was optimized. This constitutes a kind of worst case analysis.

Several researchers have studied similar checkpointing problems in the context of fault-tolerant processing sys-

*Copyright ©2005 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising and promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

[2]Update operations are those that change the state of the server object.

tems. Most results are obtained for settings where a long-running job performing heavy calculations, is checkpointed from time-to-time [15, 7, 8, 3, 9]. Other models include request processing or message logging systems [5, 13, 10]. Two works that use the average response time as optimization criterion are [6] and [5]. Both works are different from ours in that the authors consider failover time to be proportional to the checkpointing interval. A recent survey by Elnozahy et al. [4] provides an excellent overview of rollback-recovery protocols. However, analysis of optimal checkpointing interval in different contexts is not covered by the references therein.

To our knowledge this is the first time a mathematical model of a logging and failover mechanism is done at such a detailed level, i.e. including backlog processing, aiming to define the checkpointing interval that minimizes average response time.

2 Background and scope

Our mathematical model is based on an FT-CORBA infrastructure extensively used in earlier empirical studies [12]. This paper generalizes the approach to similar infrastructures. The model encompasses the logging and checkpointing unit, plus part of the application (the server object). The work in this paper aims at supporting the tuning of one parameter (the checkpointing interval) in order to achieve minimal average response time.

In the FT-CORBA infrastructure, the logging and checkpointing unit is assumed not to fail . In addition, the work assumes that there are enough backups available for the given failure frequency. If this assumption does not hold in some context then the analysis in this paper has to be extended to include the repair time of the failed primary.

2.1 The checkpointing and logging procedure

Note that throughout this paper, capital letters (e.g. P,R_T) designate random variables, and $E[P]$ designates the average value of P.

Figure 1 shows the general scheme of checkpointing and logging procedures. Upon arrival from the client, update method calls are logged by a `Logging server`, if no reply to the call is found in the log. Reply information (when present) is used by the middleware to avoid reexecuting a call, when a client resends it due to a late response. The main server that handles the application object is denoted by `Application server`. Thus, when no reply is present in the log, the request proceeds to this server for being processed on the object and accomplishing the actual serving of the client's request. After this, the request "returns" to the `Logging server` to log its reply. Here, calls are executed sequentially in a run to completion manner (a reply logging request waits for a call logging request to finish, and vice-versa). For state consistency reasons, the `Application server` also processes requests in a run

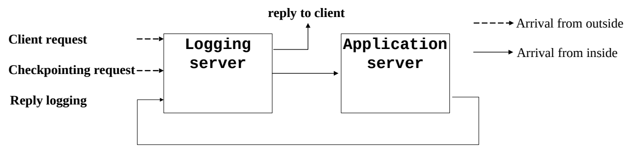


Figure 1: The logging and application servers and their relation

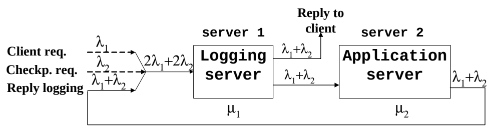


Figure 2: Servers and queues

to completion manner, in the same order that call information was logged.

From time to time the infrastructure initiates a checkpointing request (`get_state` call). This call is treated in a similar manner to the calls generated by clients. The checkpointing procedure has six phases:

Upon arrival of a `get_state` request, the last call record from the log has to be marked. This is needed in order to know which call records to remove from the log after recording the state. Thus, the call to `get_state` first spends some time in the queue of the `Logging server` , and then effectively performs the marking . Next, the call to `get_state` waits for all update calls in the queue of the `Application server` to finish execution on the object. After this, the state of the object, as left by the last update operation executed before `get_state`, is read. Finally, the state is recorded at the `Logging server`, after queuing behind calls at that server.

The above six phases are repeated P (the checkpointing interval) time units after the last phase was completed.

Figure 2 depicts some variables in our model: request arrival rates (e.g. λ_1) and service rates (e.g. μ_1). In terms of queuing theory, we model the ensemble given by the logging and application servers as a set of two servers (`server 1`, i.e. the `Logging server` and `server 2`, i.e. the `Application server`). Customers of `server 1` are client calls as well as checkpointing requests, so-called "external" customers. These customers, when departing from `server 1`, become customers of `server 2`. Customers of `server 2` when departing, return to `server 1` as "internal" customers.

2.2 Modelling assumptions

In line with earlier works [13, 5, 15, 7, 3] we assume that a failure is detected as soon as it occurs. We also assume that no failures occur during a failover [15, 5, 3].

To simplify computations we will approximate the interarrival time between two consecutive checkpointing requests with independent identically distributed random variables, with exponential distribution with average $\frac{1}{\lambda_2}$.

We assume that the probability distribution of service times on the two servers does not depend on the type of request the server is processing. This means, for example, that the probability distribution for the service time of a call logging request on `server 1` is the same as the distribution for the service time of a reply logging request. The state transfer part of the failover time is assumed to be a constant (similar to [5, 13]).

Client request ("external" customer) interarrival times are independent identically distributed variables with exponential distribution. Service times on the two servers and call replay time are also exponentially distributed. By Burke's theorem ([2]) it follows that interdeparture times are also exponentially distributed. This implies that "internal" customer interarrival time distributions are also exponential. As all interarrival time and service time distributions for the two queues are exponentially distributed, the two queuing systems are $M/M/1$ [1]. We assume that no infinite queues build up at any of the servers, i.e. the relations $\lambda_1 + \lambda_2 < \mu_2$ and $2\lambda_1 + 2\lambda_2 < \mu_1$ hold between λ_1, λ_2, μ_1, and μ_2. Also, we assume that the service rate on the `server 1` is much larger than the service rate on `server 2`, i.e. $\mu_1 >> \mu_2$.

The failure interarrival time distribution is also exponential (similar to [15, 9, 7, 5]). The average failure rate is λ_f. We also assume that the first failure arrives after the first checkpoint operation was completed.

We assume that reply logging "customers" of `server 1` have higher priority than other customers (client request or checkpointing). As a consequence, the failover time consists of (1) state transfer time (denoted by a constant), (2) wait time for all reply logging customers present in `server 1`'s queue to be served, before the call information log can be read for the replay, (3) replay time of calls corresponding to requests arrived after the last checkpoint.

In the current work we assume that *no* client request arriving at `server 1` has been resent by the client, i.e.is one that finds a reply in the call log.

3 Backlog and the average response time

As shown in Figure 3, the time line $(0, \infty)$ will be sliced in groups of failover (F), backlog processing (B) and *equilibrium* intervals. Equilibrium corresponds to normal processing. The backlog processing is the interval during which the server system (mainly `server 2`, as its service rate is the lowest) has to process all requests queued up during failover. During this time, `server 2` is busier with request processing and more prone to failures. Therefore, the average rate of failures during this time will be larger than the "equilibrium" failure rate.

3.1 Expressing the average response time

The average response time is the average time a request spends in the system given by `server 1` and `server 2`, from its arrival in the queue of `server 1` as call logging request, until it leaves `server 1` as reply to the client.

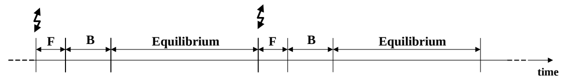


Figure 3: Equilibrium, failover and backlog processing

The server system's life cycle can be divided in two types of phases: failover when the response flow is stopped, and equilibrium and backlog processing, when the system is processing requests. Hence, the average response time is basically given by two parts: the average time spent when actual processing of the request takes place plus the average extra time spent in the queue of some server due to failover.

Details about the computation of the formula for the upper bound on the average response time (used in the minimization) with all its four terms and the formula of the checkpointing interval (p) as function of λ_2 can be found in [11].

An abstraction of the four terms present in the formula used to minimize the average response time is given below.

- Term 1 expresses the average processing and wait time spent by requests that arrive and leave during equilibrium, weighted by the fraction of time the system is in equilibrium while enough time is left for a request to finish execution on `server 2`
- Term 2 expresses the average processing and wait time spent by requests arrived during backlog processing, weighted by the fraction of time the system spends in backlog processing
- Term 3 expresses the average processing, wait and extra wait time spent by requests that arrive while the system is executing a failover (before backlog processing), weighted by the fraction of time the system is executing failover
- Term 4 expresses the average processing, wait and extra wait time spent by requests that arrived during equilibrium, but did not have enough time to complete their execution on `server 2`, weighted by the fraction of time left of equilibrium till the failure occurrence, during which there is no time to leave `server 2`

We use the tool Mathematica [14] to perform the minimization of the average response time ($E[R_T]$). The value of p that minimizes the average response time, can be computed by substituting the obtained value for λ_2 in the equation that expresses the checkpointing interval. In the next section we will present interesting relations between the functions involved in our analytical studies (e.g. λ_2 that minimizes $E[R_T]$, minimal $E[R_T]$, etc.) and some of the system parameters λ_1, λ_f, μ_2, μ_1 and μ_3.

4 Model analysis

In this section we show how studies of the model using Mathematica provided insights about the optimized variables and the input parameters of the model.

We begin by fixing some application dependent parameters as expected to be done by the system architect. In the series of studies that we will show below, μ_1 (average service rate on `server 1`), μ_2 (average service rate on `server 2`) and μ_3 (average call replay rate) are fixed at values $\mu_1 = 50$, $\mu_2 = 5$, and $\mu_3 = 12$. The choice of the two values 50 and 5 was based on the need to impose a large ratio between the average service rate on the `Logging server` and on the `Application server`. Also, the choice of the value of μ_3 such that $\mu_3 > \mu_2$ was based on the assumption that replay does not involve middleware related overhead, and thus happens faster.

Among input parameters is also $\mathtt{s}$, the time taken for state transfer (constant). We performed experiments using several values of $\mathtt{s}$. We noticed that by varying $\mathtt{s}$ the obtained curves kept the same shape, while only moving on one of the axes with constant values. All figures in the next sections use $\mathtt{s} = 0$.

We expected the studies to help us to:

- find out the behavior of the minimum average response time when considered as a function of λ_1, and how the average response time in the absence of failures and checkpointing, relates to this minimum.
- find out the relation between λ_2 that minimizes the average response time, when considering it as a function of λ_1 with a fixed (assumed) $\lambda_{\mathtt{f}}$.

4.1 Relating average response time to arrival rates

This section reports on two studies on the average response time as function of the load (λ_1). The first one arrives at the minimal average response time for various failure rates including failure rate zero (note that when $\lambda_{\mathtt{f}} = 0$ no checkpointing is needed). The second analysis covers sensitivity of the average response time to load variations. Given that the system is configured to checkpoint so that average response time is minimized, we would like to see the variations in the average response time if the actual load is very different from the assumed load.

Figure 4(a) summarizes the somewhat unexpected results on the relation between the minimum average response time and failure rate. The curves show that the response time is highly dependent on the load after arriving at a certain load (in this study ≈ 4, quite close to the limit $\mu_2 = 5$). The unexpected phenomena is that the curves are so close to each other for most values of λ_1. That is, the minimal average response times for very different average failure rates are almost identical.

One would expect that the curve for the average response time when no failures occur is much lower than the other four curves. We can see, however, that this curve always stays below the others, but is very close to them.

The graph in Figure 4(b) presents the case when the system is configured to give the minimum average response time for a given load ($\lambda_1 = 1$ and $\lambda_1 = 2.25$ respectively). Thus, the system is configured to checkpoint with a rate λ_2 that minimizes the average response time for the respective loads at a given failure rate ($\lambda_{\mathtt{f}} = 0.01$). We see that as the load is increased the average response time does not vary that much (almost constant) for both configurations until the load reaches a limit. We also see that the two configurations are behaving similarly below their respective limits. After the load reaches this limit (for one configuration around four times the assumed load, and for the second configuration twice the assumed load) average response time diverges. This is due to increase in failover time, which is now so large that the value of one element in the formula of the average response time cannot be computed, due to the expression in the logarithm present therein becoming negative.

A dual study (not shown here), varying failure rates, shows similar results. Somewhat expected, if the failure rate is low ($\lambda_{\mathtt{f}} = 0.00001$), it takes a large variation in the load to reach the worst-case average response time (more than twice the assumed load), compared with the case where the failure rate is high, $\lambda_{\mathtt{f}} = 0.01$, and the worst-case average response time is reached earlier. However, before this extreme load the response times are similar.

4.2 Relating checkpoint arrival rate to request arrival rates

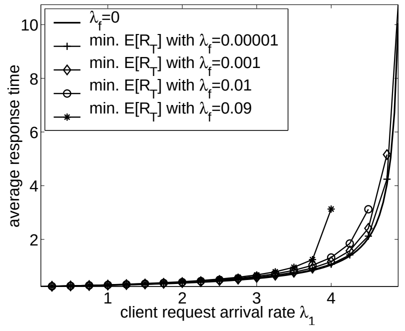
This section will present the checkpointing rate λ_2 that minimizes the average response time ($\mathtt{E}[\mathtt{R_T}]$), as a function of λ_1, in presence of certain failure patterns.

One may ask why is it interesting to study checkpointing rate as a function of λ_1? Because we wanted to find out how increasing load is handled by the system in presence of checkpointing requests and failures, and how the average checkpoint rate should behave to obtain minimal average response time.

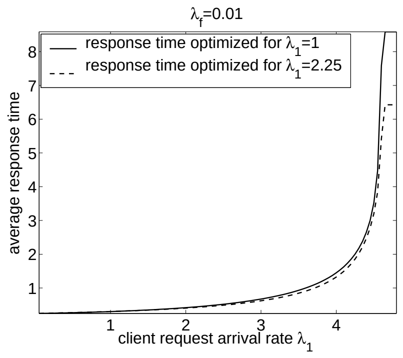
The graph in Figure 5 presents the variation of the value of λ_2 that minimizes the average response time, when considered as a function of λ_1, and four different failure rates ($\lambda_{\mathtt{f}}$). We found that the behavior of λ_2 is approximately independent of the failure rate. Until a certain value of λ_1 (≈ 2.75), the value of λ_2 increases and then it starts decreasing. The variation and the magnitude of λ_2 is, however, different from one failure rate to another. For $\lambda_{\mathtt{f}} = 0.00001$, the variation is between $0 - 0.025$; for $\lambda_{\mathtt{f}} = 0.01$ the variation is between $0.025 - 0.15$. For a quite high failure rate ($\lambda_{\mathtt{f}} = 0.09$) the value of λ_2 is almost constantly increasing. Thus the magnitude and the variation are both larger.

5 Conclusion

In this paper we proposed a detailed mathematical model of a primary-backup replicated server with logging of client calls. We used this model to study the average response



(a) Minimum $\mathbf{E}[\mathbf{R_T}]$ for several λ_fs



(b) $\mathbf{E}[\mathbf{R_T}]$ for different λ_1s and minimizing λ_2 configured assuming $\lambda_f = 0.01$

Figure 4: Average response time (y axis) plotted against λ_1 (x axis)

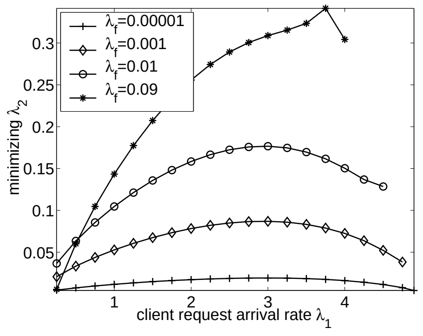


Figure 5: Minimizing λ_2 for various λ_fs plotted against λ_1 (x axis)

time as optimization criterion. An element crucial for the minimization was elaborated: the backlog processing.

The type of analysis presented can also be used to study e.g. the 99th percentile response time. This might give an insight on what are the worst-case response time behaviors of a system, given the (minimal) average response time. While these figures can easily be approximated (by an upper bound), by using Markov's inequality [1], we believe that they are too pessimistic to be useful in analysis of distributed service-oriented architectures. Rather, we propose the sensitivity analysis with respect to the load for which the system was configured for, as demonstrated in the paper.

In this work we looked at the failover time in terms of probability distribution of the number of calls to replay, while assuming the distribution of replay time to be independent of the call to replay. However, to deal with more complex systems, where different calls may have different replay times, an immediate extension to the model is to consider different probability distributions depending on the call to replay. Similar arguments apply to the assumption about the state transfer time considered a constant here. For more complex applications we may need to use a random variable to model state transfer time. These are directions for future extensions of this work.

Acknowledgments

This work has been supported by CENIIT (CENter for Industrial Information Technology) at Linköping University and by the FP6 IST project DeDiSys on Dependable Distributed Systems.

References

[1] A.O. Allen. Probability, Statistics, and Queueing Theory with Computer Science Applications. Academic Press, 1978.

[2] P.J. Burke. The Output of a Queueing System. Operations Research, 4:699–704, 1956.

[3] E.G. Coffman, L. Flatto, and P.E. Wright. A Stochastic Checkpoint Optimization Problem. SIAM Journal on Computing, 22(3):650–659, 1993.

[4] E.N. Elnozahy, L. Alvisi, Y.–M. Wang, and D. B. Johnson. A Survey of Rollback–Recovery Protocols in Message-Passing Systems. ACMComputing Surveys, 34(3):375–408, September 2002.

[5] E. Gelenbe and D. Derochette. Performance of Rollback–Recovery Systems Under Intermittent Failures. Communications of the ACM, 21(6):493–499, June 1978.

[6] Y. Huang and P. Jalote. Effect of Fault Tolerance on Response Time - Analysis of the Primary Site Approach. IEEE Transactions on Computers, 41(4):420–428, April 1992.

[7] C.M. Krishna, K.G. Shin, and Y.–H. Lee. Optimization Criteria for Checkpoint Placement. Communications of the ACM, 27(10):1008–1012, October 1984.

[8] V.G. Kulkarni, V.F. Nicola, and K.S. Trivedi. Effects of Checkpointing And Queueing on Program Performance. Communications on Statistics–Stochastic Models, 6(4):615–648, 1990.

[9] J.S. Plank and M.G. Thomason. The Average Availability of Uniprocessor Checkpointing Systems, Revisited. Technical report, Department of Computer Science University of Tennessee, August 1998.

[10] K.–F. Ssu, B.Yao, and Fuchs W.K. An Adaptive Checkpointing Protocol to Bound Recovery Time With Message Logging. In Proceedings of the Symposium on Reliable Distributed Systems, pages 244–252, 1999.

[11] D. Szentiványi. Performance Studies of Fault-Tolerant Middleware. PhD thesis, Linköping University, March 2005. Available at http://www.ep.liu.se/diss/science technology/09/29/index.html.

[12] D. Szentiványi and S. Nadjm-Tehrani. Middleware Support for Fault Tolerance. In Qusay H. Mahmoud, editor, Middleware for Communications, chapter 18. Wiley & Sons, 2004.

[13] A. N. Tantawi and M. Ruschitzka. Performance Analysis of Checkpointing Strategies. ACM Transactions on Computer Systems, 2(2):123–144, May 1984.

[14] Inc. Wolfram Research. Mathematica. http://www.wolfram.com/products/mathematica/, November 2004.

[15] J. W. Young. A First Order Approximation to the Optimum Checkpointing Interval. Communications of the ACM, 17(9):530–531, September 1974.