Optimal Choice of Checkpointing Interval for High Availability

Diana Szentiványi and 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

Supporting high availability by checkpointing and switching to a backup upon failure of a primary has a cost. Trade-off studies help system architects to decide whether higher availability at the cost of higher response time is to strive for. The decision will lead to configuring a fault-tolerant server for best performance. This paper provides a mathematical model employing queuing theory that helps to compute the optimal checkpointing interval for a primary-backup replicated server. The optimization criterion is system availability. The model guides towards the checkpointing interval that is short enough to give low failover time, but long enough to utilize most of the system resources for servicing client requests. The novelty of the work is the detailed modelling of service times, wait times for earlier calls in the queue, and priority of checkpointing calls over client calls within the queues. Studies on the model in Mathematica and validation of a modelling assumption through simulations are included.

1 Introduction

High availability is a core requirement of many services on which the society depends. A great challenge for the system architect is how to perform the availability-performance trade-off in the light of changing service contexts. Today's open systems have varying workloads and complex structure. The failure of one component may well lead to collapse of a whole range of services due to uninformed decisions made during system configuration and dimensioning. The vision for our work is to build up a tool set that supports the engineer in adaptation of a networked service to changing circumstances. We propose modelling and analysis of mechanisms for dealing with failures and maintaining an acceptable service while the system is recovering from the results of a server crash. In particular, we illustrate the application of this approach to mechanisms that are embedded in every middleware that supports delivery of fault-tolerant services. One example is a CORBA-based infrastructure in which primary-backup replication mechanism is one way for the system architect to implement fault-tolerant services [14].

A common problem faced by system developers that aim to use automatic support for fault detection and failover is the complexity of the platforms that are suggested for this purpose. There are simply too many parameters that can affect the performance of a system and setting/changing one parameter in one way or the other is deemed to have unpredictable influence in the range of scenarios that the system will face.

In cold primary-backup replication style that we consider in this paper, a backup is only started when the primary fails. Furthermore, the state of the primary has to be checkpointed *periodically* to some local storage. With frequent checkpointing the time required to switch the service to a backup server, also called *failover time*, is lower because fewer calls need to be replayed on the backup server. However, during checkpointing, the system stops serving update client requests. With less frequent checkpointing the system has a better steady state behaviour at the expense of a longer failover time. Normally the system architect has no systematic means of finding the optimum in this trade-off. This paper applies our modelling and analysis methodology to derive optimal checkpointing intervals for a given replicated service. As well as a theoretic ground for analyzing the above trade-off, one can see the approach as a basis for implementing a systematic tool for adaptation of a system to changing circumstances.

The contributions of the paper are as follows. We propose a detailed model that includes an application server and a server that supports the failover mechanisms by logging the client requests and the computed replies. The model includes waiting times in both servers' queues using queuing theory, as well as both servers' service times. It also characterizes the available intervals of the application server both as intervals of steady state and intervals in which backlog processing due to failover from a primary takes place. The paper illustrates the application of the model to compute an optimal checkpointing interval that maximizes the average system availability. This optimal value is then tested in a simulation model of a system in which the checkpointing interval is considered as a constant (as in the real world). We show that the mathematical computations yield results that are good enough to be used in a constant interval setting. To place the work in perspective, we recall that the average availability as optimization criterion is typically not suitable for hard real-time applications [7]. However, we consider the model powerful enough for meaningful analysis in a wide range of high-availability applications. This can be confirmed by other researchers that use a similar criterion (Plank et al. [10] and Vaidya [17]). A recent survey by Elnozahy et al. [5] provides an excellent overview of rollback-recovery protocols. However, analysis of optimal checkpointing interval in different contexts is not covered by the references therein.

Several researchers studied the above trade-off, i.e. the problem of the "optimum checkpointing interval" in the context of fault-tolerant *processing systems*, in settings where a long-running job (performing heavy calculations), or a process, is from time-to-time checkpointed [19, 7, 8, 4, 10, 9]. Other models include request processing or message logging systems [6, 16, 12] and request processing in mobile environments [3]. Our analysis fits in this second category. However, we provide a more precise optimum by explicitly modelling number of replayed calls as part of the failover time.

Next we place the work in the context of own earlier works. In the empirical studies of an implemented platform that was tested with a telecom service, the checkpointing interval was fixed ad hoc and the resulting middleware overhead was emphasized [14]. Later, a basic framework was suggested in which checkpointing interval was optimized for finding the shortest *average response time*. In that model the checkpointing request was treated in the same way as other calls at the logging and application servers [15]. This work extends that approach by using priority queues and considering average availability as optimizing criterion.

2 Background and scope

In this paper, we model a proposed implementation of a standard-compliant FT-CORBA infrastructure [14]in which the logging and checkpointing infrastructure unit is assumed not to fail [1]. This means that information stored to enable a failover will never be lost.

Figure 1 shows the checkpointing and logging procedures as they take place in a standard compliant FT-CORBA infrastructure . The `Logging server` updates a state log and a call log. The main server that handles the application object is denoted by `Application server`. The state log contains information about the last recorded state. Upon arrival from the client, the calls to *update* operations

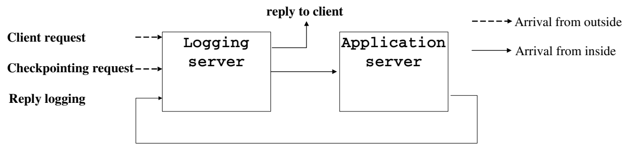


Figure 1: The logging and application servers

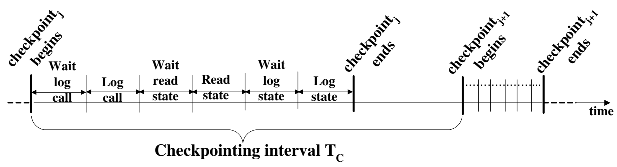


Figure 2: The checkpointing procedure time slices

are logged as records containing enough information (e.g. operation name and parameters) to be able to replay them on the backup if needed. Update operations are those that change the state of the server object (in contrast with read-only operations that leave the object state unchanged after execution). Each time a new state record is saved in the state log, the records corresponding to calls that were executed on the object before the current state reading took place, are removed from the call log. The log also includes information about the result of executing a client query on the application server. This is used by the middleware to avoid multiple executions of the query. In the initial model we assume that no client request is resent. Extending the model to multiple calls from the client is straightforward.

We analyze the primary-backup service availability as if there were sufficient number of backups available for the given failure frequency. A special case of this is the assumption of one backup, and that there will be no failures until the state of the recently failed primary has been reconstructed. If the number of backups is limited or the latter assumption is not valid in the circumstances then the analysis in this paper has to be extended to include the repair time for a server.

After recording its call information in the log, an update request proceeds to the `Application server` for being processed on the object and accomplishing the actual serving of the client's request. After this, it "returns" to the `Logging server` to write its reply in the log. From time to time the infrastructure initiates a checkpointing request (by initiating a `get_state` call on the `Application server`). A `get_state` operation arriving at the `Application server`, despite being a read operation, has to wait until the currently executing update call finishes execution and leaves the server in a "frozen" state. This means that the call to `get_state` is treated in a non-preemptive prioritized way.

To summarize, there are two categories of call information logging: (1) calls corresponding to the client re-

[1] To combine the effects of infrastructure failure and application failure it is also possible to use a non-standard (fully available) FA-CORBA infrastructure [14] but that is outside the scope of this paper.

quests and (2) calls corresponding to the get_state request. There are two categories of reply logging: (1) the returned result by the client initiated calls, and (2) the result of the get_state call, i.e. the completed checkpointing of the state. In all cases, the calls are executed sequentially in a run to completion manner at the Logging server (i.e. no interruptions of one execution by another). However, reply logging operations are prioritized over the call logging operations. Thus, our model includes extra wait times for call logging operations. For state consistency reasons, the Application server also processes requests in a run to completion manner in the same order that call information was logged.

3 The checkpointing procedure

The checkpointing procedure has six phases: (1) waiting to log the get_state call record, (2) logging the get_state call record, (3) waiting to execute the "state reading" on the Application server, (4) reading the state from the server, (5) waiting to record the state, (6) recording the state in the Logging server. In what follows we explain the above in more detail.

Upon arrival of a get_state request, it is logged at the Logging server. This is needed to know which calls arrived before this get_state request. Out of them, those executed at the Application server before get_state have to be removed from the call log once the result of the get_state is recorded.

Thus, the call to get_state spends some time waiting for and receiving service at the Logging server (phase 1 and 2 above, denoted by "Wait log call" and "Log call" in Figure 2).

Next, the call to get_state arrives at the Application server, and waits for the currently executing update call to finish execution on the object. After this, the get_state request is executed on the application object: the state of the object (as left by the update operation executed before get_state) is read (phases 3 and 4 above, denoted by "Wait read state" and "Read state" in Figure 2).

Finally, the state that was read by get_state is recorded at the Logging server, after waiting for all reply logging calls that are in the queue in that server. Thus, before "exiting the system", the completed get_state request again spends some time in the queue of the Logging server and on it (phases 5 and 6 above, denoted by "Wait log state" and "Log state" in Figure 2).

The six phases constitute the whole checkpointing procedure, and are repeated every T_C (checkpointing interval) time units. Failures can occur even during the six phases of checkpointing mentioned above.

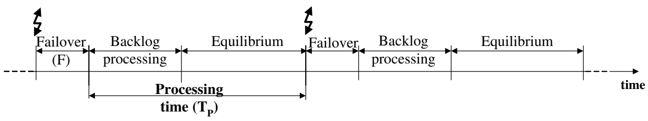


Figure 3: Failover, backlog processing, and equilibrium

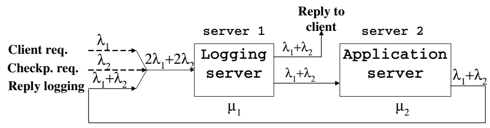


Figure 4: Servers and queues

4 The model

The aim of the model is to express the average availability for client request processing, as a function of the "unknown" quantity that is the average checkpointing interval, and the rest of "known" system parameters. Once this function is found, the main goal of finding the (average) checkpointing interval that maximizes availability, can be achieved.

Throughout this paper, capital letters (with or without subscript, e.g. S_A, F) designate random variables, and $E[F]$ designates the average value of F.

As shown in Figure 3, the time line $(0, \infty)$ will be sliced in groups of failover (F), backlog processing and *equilibrium* intervals. Equilibrium corresponds to normal processing. 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.

Following the description in Section 2, we model the server side of the application (see the two servers in Figure 1) as a Jackson network [1]. The first server, where client and checkpointing requests perform call (and later reply) information logging is named server 1 in Figure 4. The second server at which requests arrive for processing on the application object is named server 2 in Figure 4. The two servers form a pipeline. Call logging (i.e. external) customers of server 1, when departing from it, become customers of server 2. Customers of server 2 when departing, return to server 1 as reply logging customers. From server 1 they depart as "reply to client". The service rates of the two servers - μ_1 and μ_2 - are shown in Figure 4 under the respective boxes. The customer arrival rates - λ_1 for client requests and λ_2 for checkpointing requests - are shown on the respective flows. Parameter μ_3 (not shown in the picture) will be used as the call replay rate.

In this model, at server 1 we have three types of customers, each of them having its own priority. At server 2 we have two types of customers with two priorities. These two priority queues in the model suffice to implement the non-preemptive prioritized policy mentioned earlier.

4.1 Modelling assumptions

As in earlier models [16, 6, 19, 7, 4, 9] we assume that a failure is detected as soon as it occurs. We also assume that no failure occurs during a failover [19, 6, 9, 4]. The first failure arrives after the first checkpoint operation was completed. Failure interarrival time distribution is exponential, as in [19, 10, 17, 7, 6, 3].

We assume that checkpointing request interarrival times are independent identically distributed random variables, with exponential distribution (a). Also, client request interarrival times are independent identically distributed variables with exponential distribution (b). Service times on the two servers and call replay time are also exponentially distributed (c). From (a), (b), and (c) and Burke's law [2] it follows that the internal customers of the two servers have also exponentially distributed interarrival times. Related to queue analysis, we assume that no infinite queues build up at any of the servers. This means that the following relations hold: $\lambda_1 + \lambda_2 < \mu_2$ and $2\lambda_1 + 2\lambda_2 < \mu_1$. Also, we assume that the service rate on `server 1` is much larger than the service rate on `server 2`, i.e. $\mu_1 >> \mu_2$. The average arrival rate of client requests is assumed to be larger than the average arrival rate of checkpointing requests, i.e. $\lambda_1 > \lambda_2$.

A simplifying assumption we make is 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 information logging request on `server 1` is the same as the distribution for the service time of a reply logging request. Also, the state transfer part of the failover time is assumed to be a constant.

We assume that *all* client requests arriving at `server 1`, will proceed to the queue of `server 2` and will be processed on `server 2`.

To be able to use similar terms in the formulas that model backlog and equilibrium respectively, two different average failure arrival rates are considered, one for the equilibrium interval and one for the backlog interval. We assume a failure arrival rate $\lambda_{\mathtt{f}}$ for the system during equilibrium and derive a backlog failure rate from that.

4.2 The queues

There are two priority queues in our model: one at `server 1`, and one at `server 2`. As all interarrival time and service time distributions for the two queues are exponentially distributed (see Section 4.1), the two queuing systems are `M/M/1` [1].

The total arrival rate for customers at `server 1` is $2\lambda_1 + 2\lambda_2$. It is the sum of the arrival rates of all types of customers in the queue (see Figure 4): (1) customers denoted by client requests, having lowest priority `1`, (2) customers denoted by checkpointing requests, having priority `2`, (3) reply and state logging customers, denoted by "feedback" of client requests from `server 2`, having highest priority `3`.

The priority queue at `server 2` contains customers that departed from `server 1` and that will eventually return to `server 1`. Hence, the total average arrival rate of customers in the queue at `server 2` is $\lambda_2 + \lambda_1$. Customers with highest priority (checkpointing requests) have average arrival rate λ_2, while customers with lowest priority (client requests) have average arrival rate λ_1.

4.3 Optimal checkpointing interval

In this section we will present the aspects involved in the computation of the checkpointing interval that optimizes the average availability. Table 1 summarises some random variables used in our optimization analysis (see also Figure 3 and 4).

The formula of the lower bound on average availability used to maximize the average availability is computed as follows: one has to divide the total time the system is available for request processing, by the total time between two consecutive failures occurrng during equilibrium (term $\mathtt{E}[\mathtt{T_P}] + \mathtt{E}[\mathtt{F}]$). The denominator's value is obtained simply by subtracting the total time spent in failover due to failures occurring during backlog processing (term $\mathtt{E}[\mathtt{N_F}]\mathtt{E}[\mathtt{F}]$) and the total time spent in processing checkpointing requests on the two servers (term $\mathtt{E}[\mathtt{N_C}](2\mathtt{E}[\mathtt{S_L}] + \mathtt{E}[\mathtt{S_A}])$), from the average processing time ($\mathtt{E}[\mathtt{T_P}]$). $\mathtt{E}[\mathtt{N_C}]$ is the average number of checkpoint operations occuring during the "availability" period of the server. Therefore $\mathtt{E}[\mathtt{N_C}] = \frac{\mathtt{E}[\mathtt{T_P}] - \mathtt{E}[\mathtt{N_F}]\mathtt{E}[\mathtt{F}]}{\mathtt{E}[\mathtt{T_C}]}$.We describe the formula below:

$$\begin{aligned}
\mathtt{E}[\mathtt{A}] &= \frac{\mathtt{E}[\mathtt{T_P}] - \mathtt{E}[\mathtt{N_F}]\mathtt{E}[\mathtt{F}] - \frac{\mathtt{E}[\mathtt{T_P}] - \mathtt{E}[\mathtt{N_F}]\mathtt{E}[\mathtt{F}]}{\mathtt{E}[\mathtt{T_C}]}(2\mathtt{E}[\mathtt{S_L}] + \mathtt{E}[\mathtt{S_A}])}{\mathtt{E}[\mathtt{T_P}] + \mathtt{E}[\mathtt{F}]} \\
&= \frac{(\mathtt{E}[\mathtt{T_P}] - \mathtt{E}[\mathtt{N_F}]\mathtt{E}[\mathtt{F}])(1 - \frac{2\mathtt{E}[\mathtt{S_L}] + \mathtt{E}[\mathtt{S_A}]}{\mathtt{E}[\mathtt{T_C}]})}{\mathtt{E}[\mathtt{T_P}] + \mathtt{E}[\mathtt{F}]} \\
&= \frac{(\mathtt{E}[\mathtt{T_P}] - \mathtt{E}[\mathtt{N_F}]\mathtt{E}[\mathtt{F}])\left(1 - \lambda_2\left(\frac{2}{\mu_1} + \frac{1}{\mu_2}\right)\right)}{\mathtt{E}[\mathtt{T_P}] + \mathtt{E}[\mathtt{F}]}
\end{aligned}$$

We need to compute the following values: the average failover time ($\mathtt{E}[\mathtt{F}]$), the average (non-zero) number of failures that occur during backlog processing ($\mathtt{E}[\mathtt{N_F}]$), and the average processing time from the end of a failover until the moment of the next failure ($\mathtt{E}[\mathtt{T_P}]$). More detailed information on these rather complex computations that we omit here due to lack of space, can be found in Chapter 7 of [13]. In the present work we have added the effect of priority queues that influences (1) the wait times in the queues of the two servers, and (2) the way to compute the average number of requests to replay at failover.

After the above computations $\mathtt{E}[\mathtt{A}]$ is obtained as a function of seven parameters: μ_1 (`server 1` service rate), μ_2 (`server 2` service rate), λ_1 (average load), λ_2 (checkpoint arrival rate), $\lambda_{\mathtt{f}}$ (failure rate), μ_3 (call replay rate), `s` (state transfer time). Considering all parameters beside λ_2, as so-called constants, we obtain $\mathtt{E}[\mathtt{A}]$ as a function `f` in only one unknown. To obtain the optimal checkpointing

A	availability	–
T_P	processing time from the end of failover i until failure $i+1$	–
N_F	number of failures during the backlog processing interval	–
F	failover time	–
T_C	time between two checkpoint request arrivals	$E[T_C] = \frac{1}{\lambda_2}$
S_L	service (processing) time on the `Logging server`	$E[S_L] = \frac{1}{\mu_1}$
S_A	service (processing) time on the `Application server`	$E[S_A] = \frac{1}{\mu_2}$
s	state transfer time	s appears in the formula of the average failover time E[F]

Table 1: Variables used in the model

interval it remains to maximize $f(\lambda_2)$. In the rest of this paper the Mathematica tool [18] has been used to numerically perform maximization.

5 Numerical studies

This section summarizes our studies using the presented model. The goals of the studies were:

1. to find out the behaviour of the highest average availability when considered as a function of the load (λ_1), respectively failure rate (λ_f), and the extent (especially the scale) of the dependency. We also wanted to see how a fixed λ_f, respectively λ_1 influenced the studied behaviours.
2. to provide guidelines to the system architect in determining the optimal checkpointing interval given a fixed λ_1 and λ_f. For a given load, when considering λ_2 as a function of failure rate, we expect to find out the critical failure rate that invalidates the desired relation "(on average) checkpointing is done more often than failures occur" (i.e. the point at which $\lambda_2 > \lambda_f$ no longer holds).

In the series of studies that we will show below, μ_1, μ_2 (average service rates on `server 1` and `server 2`), and μ_3 (average call replay rate) are fixed at values $\mu_1 = 50$, $\mu_2 = 5$, and $\mu_3 = 12$. With these infrastructure/application related measures fixed, it is meaningful to study how the system handles different external conditions such as different loads (λ_1), and different average failure rates (λ_f). The choice of the first two values (50 and 5) was based on the need to impose a large ratio between the two server's service rates. 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. The studies included experiments for different values of s (state transfer time). Our observations confirmed our expectations: by varying 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 $s = 0$.

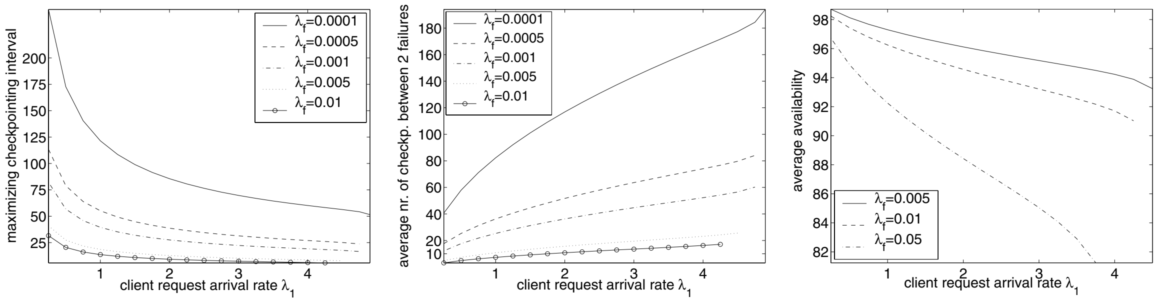
5.1 Relating checkpointing interval to load

Figure 5(a) shows the dependency of the maximizing average checkpointing interval ($\frac{1}{\lambda_2}$) on the chosen failure rate and its variation with load. The maximizing $\frac{1}{\lambda_2}$ presents similar behaviour when the load grows, independent of failure rates. The value of the checkpointing interval decreases as load grows. However, the variation and the magnitude of the maximizing checkpointing interval is different from one failure rate to another. For example, when $\lambda_f = 0.0005$, the variation is approximately between $30 - 120$ (time units); for $\lambda_f = 0.01$ the variation is between $6 - 30$. This behaviour is somewhat expected. It is interesting to see that as the failure rate increases the checkpointing interval becomes smaller and smaller (for same load) in order to maximize average availability. What is not visible in this picture is the approximate number of checkpoints that take place between two consecutive failures. The variation of this number with load and the dependency of this behaviour on the failure rate is illustrated in Figure 5(b). The picture tells us that as failures become sparser the approximate average number of checkpointing operations that occur between two failures increases. This somewhat strange phenomenon has a rather simple explanation: when the distance between failures is large, checkpointing has to be done more often in order to increase the chance of a state recording to happen shortly before the next failure.

Next, we study maximum average availability in relation to λ_1 for different failure rates. Figure 5(c) shows that the behaviour of maximum average availability (E[A]) in relation to λ_1 is similar for all chosen failure rates : it decreases, as λ_1 grows. However, for $\lambda_f = 0.005$, the decrease happens with a much smaller slope than for the larger λ_fs.

5.2 Relating checkpointing interval to failure rate

Next we try to answer the question: from which failure rate (λ_f), if any, the desired property that "on average checkpointing happens at least once between two failures" ($\lambda_2 > \lambda_f$) ceases to hold? To make this study we plot the variation of the maximizing checkpoint arrival rate (λ_2, the inverse of the checkpointing interval) against failure rate. We plot two graphs in Figure 6(a), both shedding light on this question and at the same time giving new insights about the relationship between λ_1 (load), λ_f and the maximizing λ_2. We choose two arbitrary values for λ_1 (1 and 4), one representing light load compared with the average service rate of `server 2` (1 compared to 5) and one representing high load (4 compared to 5). For each value of λ_1 we further study the dependence of another parameter, namely the average failover time, on failure rate (Figure 6(b)). Finally, we consider the effect on the maximum average availability when varying λ_f (Figure 6(c)). A summary of our insights



(a) Maximizing checkpointing interval behaviour for several failure rates

(b) Average number of checkpoints between two failures

(c) Maximum average availability for different failure rates

Figure 5: Checkpointing interval, number of checkpoints and maximum average availability (y axis) plotted against load λ_1 (x axis)

is given below:

- the average failover time is relatively large for small failure rates. The rapid decrease towards almost constant values is present for both the small and the large load. This insight supports the engineer that may choose to use our tool for analysis: is it good enough to have a few but long failovers, or would one prefer many failovers that are shorter? Note that when failures are sparse (i.e. mean time between failures is e.g. 100000 time units), even if the average availability is maximized, individual failover times can be quite large.
- for both the small and the large loads considered, the checkpointing rate that leads to maximal average availability is always larger than the failure rate. Figure 6(a) also shows that, as expected, with larger load, one needs to checkpoint more often to maximize average availability. This is due to the need to reduce the replaying element in the failover time with larger loads.
- as expected, the maximum average availability decreases as failure rate increases. The slope of the decrease is much slower in the situation when low load is applied, than when the load is large.

6 Simulations

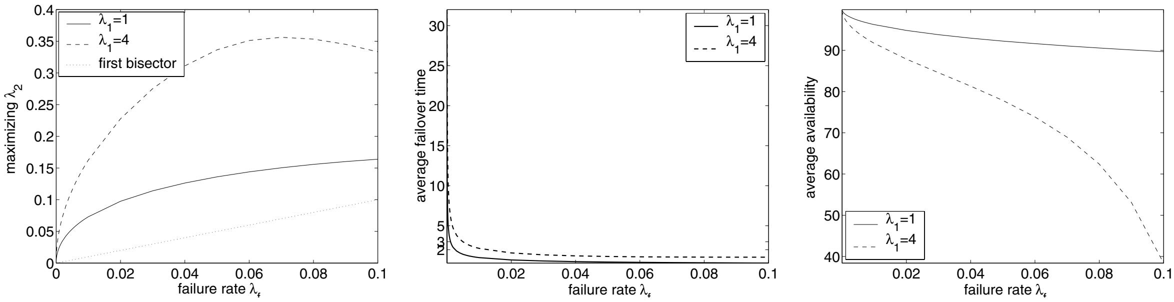
When building the mathematical model we approximated the checkpoint interarrival time with an exponentially distributed random variable. This assumption was needed to make the computation of the maximizing checkpointing interval $\frac{1}{\lambda_2}$ based on the (random variable) inputs possible. In reality, the checkpointing interval is a deterministic value chosen by the system architect and configured in the middleware. The main goal of this section is to check how significant this approximation in the mathematical model is. That is, are all the observations in Section 5 valid in a world where all parameters are as in the model except that checkpointing is done regularly? In particular, if the system architect takes the result of the mathematical computations, i.e. the computed maximizing λ_2 based on a set of fixed input parameters, does this indeed produce maximum average availability or not?

6.1 Experimental setup

A simulation model was built within the SIMULA [11] programming environment. It uses assumptions similar to those in the mathematical model, i.e. exponentially distributed request interarrival, service times and failure interarrival. However, the checkpointing interarrival time is a deterministic value here. In the simulations each checkpointing request is scheduled exactly $\frac{1}{\lambda_2}$ units after the previous one, modelling a deterministic checkpointing interval.

To check whether the simulation runs confirm our intuitive expectations, we compared the average availabilities resulting from the following sets of simulations. For each value of λ_1 we considered several checkpointing intervals: one corresponding to the (average availability) maximizing checkpointing interval obtained from our mathematical analysis, and others with values both below and above the maximizing value. The failure rate λ_f remained unchanged in all these experiments.

To do the comparisons we plotted sets of curves representing the simulated average availability as a function of load (λ_1). To confirm the validity of the approximation in the model we would expect that for all choices of checkpointing interval when $\frac{1}{\lambda_2}$ was different from the maximization result, i.e. both below and above the maximizing $\frac{1}{\lambda_2}$, the simulations would show lower average availability $E[A]$. And, even if for some of the values, average availability is higher than that obtained for the maximizing checkpointing interval, the "computed" maximum (from the Mathematica model) is not much lower than the simulated values. Hence, the approximation in the mathematical model leads



(a) Maximizing checkpoint arrival rate

(b) Average failover time

(c) Maximum average availability

Figure 6: Checkpoint arrival rate, failover time and average availability (y axis) plotted against failure rate (x axis)

to a valid result in the (simulated) reality. These will provide evidence that the mathematical studies indeed lead to results applicable in reality.

To repeat the studies in the same framework as the mathematical analyses, we began by choosing failure rate $\lambda_f = 0.001$ and performed a set of 100 experiments for each value of λ_2, and for each selected λ_1. For $\lambda_f = 0.001$ we chose 18 values for λ_1 (0.25, 0.5, etc). Thus, in total we ran 18×100 experiments for each value of λ_2. Each experiment consisted of a simulation run lasting 10000 logical time units. For each experiment we computed the time the servers are not available for request processing. This sum was subtracted from the total simulation logical time (10000) essentially computing the value of the denominator in the formula of E[A] (see Section 4.3). For each of the 100 experiments the average availability was computed by dividing this value by the length of the simulation interval (10000). Finally, for each value of λ_2 and λ_1, the average of the 100 average availability values was obtained.

6.2 Results

Figure 7 shows how the average availability of the system varied with λ_1 in the simulations, using respectively the maximizing λ_2 as obtained after the maximization of E[A] performed in Mathematica, and other values of λ_2 obtained by decreasing or increasing the maximizing checkpointing interval ($\frac{1}{\lambda_2}$). We considered decreasing the maximizing $\frac{1}{\lambda_2}$ with e.g. 10%, 50%, 75%, or 90% of its value respectively. When increasing the maximizing $\frac{1}{\lambda_2}$ we considered a fixed number of percentage values (20) that were computed based on the ratio between the maximizing λ_2 (for the considered λ_1) and λ_f (in the presented graph $\lambda_f = 0.001$). For example, when $\lambda_1 = 1$, and the maximizing λ_2 had the value 0.025 (i.e. $\frac{1}{\lambda_2} = 40$, and $\frac{\lambda_2}{\lambda_f} = 25$), we considered for simulation, values of the checkpointing interval obtained by increasing 40 with e.g. 120% or 240% of its value respectively.

The result depicted in Figure 7 gives us the following

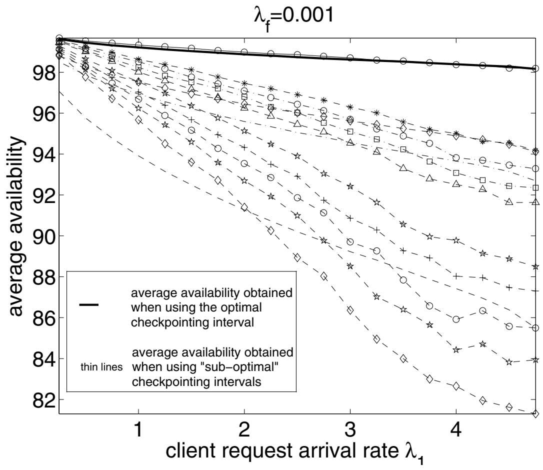


Figure 7: Average availability (y axis) plotted against λ_1 (x axis)

insight: if the system architect decides to configure the optimal checkpointing interval according to computations offered by our mathematical tool, he/she will be able to obtain values for average availability very close to a maximum. The graph in Figure 7 shows indeed that there are a few values of the checkpointing interval (always larger than the mathematically computed optimum) for which, for a given load (λ_1) and a given failure rate (λ_f), higher values for the average availability of the system can be obtained (see the solid line with circles, very close to the thick line in Figure 7). Still, one can argue that these values are much closer to the "expected" maximum than, for example, the values that are obtained when checkpointing with a smaller interval (see e.g. the dashed line triangle curve in Figure 7). Also, one could say that it is still better to checkpoint with an interval that does not always lead to the "absolute" maximum, but to a value very close to it, than to start guessing the checkpointing interval and possibly ending up in an extreme where the average availability is significantly lower than the maximal one.

All simulations of this section were repeated for $\lambda_{\mathtt{f}} = 0.01$, $\lambda_{\mathtt{f}} = 0.05$ and $\lambda_{\mathtt{f}} = 0.005$. The validity of our approximation from the mathematical model was exhibited again with similar results as in the curves already presented in this section.

7 Conclusion and future work

In this paper we proposed a mathematical model that leads to a formula of the average system availability, as a function of the checkpointing interval. Furthermore we showed how to obtain an optimal value for this important parameter of a primary-backup replicated service. By using a numerical tool, such as Mathematica, a system architect can easily feed the other system parameters such as load, failure rate, service rates, and simply obtain the checkpointing interval that is used to configure the underlying middleware leading to maximum availability.

A major result of our studies was that for a given load, with values that yield maximal average availability, the average failover time decreases as failure rate grows. Another insight was on the value of the maximizing checkpointing interval considered as a function of load: it varies according to a shape that is independent of a fixed failure rate. A counter intuitive result that has a quite simple explanation was that the average number of checkpointing requests arriving between two consecutive failures decreases as failure rate grows.

To validate our model assumption on the checkpointing interval as a random variable, we performed simulations of a system where the checkpointing interval is chosen as a constant. The outcome is that by using the mathematically computed maximizing checkpointing interval one indeed obtains an average system availability that is extremely close to an obtainable (simulated) maximum.

A major contribution of this work as compared with earlier ones is that it includes prioritized middleware operations compared to application calls, and a detailed model of backlog processing. Furthermore, it looks deeper at the failover time in terms of probability distribution of the number of calls to replay, while considering the distribution of replay time to be independent of the call to be replayed. To deal with more complex systems, an immediate extension to the model is possible where different calls may have different replay times. Similar arguments apply to the assumption about the state transfer time, considered a constant here. A constant is reasonable in the sense that one may well be able to derive such values for state transfer time in the context of each different application. However, for more complex applications we may need to vary state transfer times for different invocations, thus needing a random variable to model state transfer time. Future works include the use of such mathematically obtained optimal parameters in run-time reconfiguration of middleware.

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] X. Chen and M.R. Lyu. Performance And Effectiveness Analysis of Checkpointing in Mobile Environments. In Proceedings of the 22nd IEEE International Symposium on Reliable Distributed Systems, pages 131–140, 2003.

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

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

[6] E. Gelenbe. On the Optimum Checkpointing Interval. Journal of the ACM, 26(2):259–270, April 1979.

[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. Marcel Dekker, Inc., 1990.

[9] Y. Ling, J. Mi, and X. Lin. A Variational Calculus Approach to Optimal Checkpoint Placement. IEEE Transactions on Computers, 50(7):699–708, July 2001.

[10] 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.

[11] SIMULA Webpage. http://www.item.ntnu.no/fag/SIE5015/, August 2005.

[12] K.–F. Ssu, B.Yao, and Fuchs W.K. An Adaptive Checkpointing Protocol to Bound Recovery Time With Message Logging. In Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems, pages 244–252, 1999.

[13] 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.

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

[15] D. Szentiványi, S. Nadjm-Tehrani, and J. M. Noble. Configuring Fault-Tolerant Servers for Best Performance. In Proceedings of the 1st IEEE Workshop on High Availability of Distributed Systems (HADIS'05) - part of DEXA'05, pages 310–314, 2005.

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

[17] N.H. Vaidya. Impact of Checkpoint Latency on Overhead Ratio of a Checkpointing Scheme. IEEE Transactions on Computers, 46(8):942–947, 1997.

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

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