Run-time Distributions in Passively Replicated Systems Using Timeout and Acceptance Fault Detection

Åsmund Tjora Amund Skavhaug
Department of Engineering Cybernetics
Norwegian University of Science and Technology
O.S. Bragstads plass 2D
7491 Trondheim
Norway

Abstract

Fault tolerance based on passive replication is common in many systems. If this kind of fault tolerance mechanism is to be used in a real-time system, timing analysis is necessary, as the fault tolerance mechanism itself may cause timing faults.

There are different ways of detecting when the primary replica has failed, one of them is to use a timeout to detect crash and omission failures, another to run acceptance tests on the results, which detects some value failures. As these two detection strategies cover different kinds of failures, it can be useful to combine them.

In this paper, a mathematical model for the timing behaviour of a system with passive replication, where a combination of timeout and acceptance test is used for fault detection, is derived.

Examples are given to demonstrate how the model can be used for calculation of deadline miss probabilities, and further how this can be used for checkpoint placement optimisation.

1. Introduction

Many real-time systems will, in addition to the timing requirements, have requirements to reliability. To fullfill the reliability requirements, a fault tolerance mechanism might be necessary. While the fault tolerance mechanism may detect and correct many of the faults that occur to the system, it may also introduce new faults if the extra time used for detection and correction cause deadline misses. Because of this, the reliability improvement given by some fault tolerance mechanisms may be very small, depending on the system's real-time requirements. The effect of using fault tolerance mechainsms in real-time systems is therefore an important study, as noted in [5].

In fault tolerant real-time systems, using active replication structures, i.e. structures where a task is run on several replicas simultaneously, has been common. Even if errors are detected in some of the replicas, the non-erroneous replicas will still be able to produce results within the deadlines. In addition, comparision of the results from the different replicas can be used as fault detection, increasing the range of faults that can be detected. On the negative side, running several replicas of the same task simultaneously do require extra hardware resources, which can be costly.

Passive replication strucures are less resource consuming. In these structures, only one of the replicas running the task is active, while the other tasks are passive. If an error is detected in the active replica, it is stopped, and one of the backup replicas is prepared and made active. The task that ran on the failed replica is then rerun on the new active replica. Because of the extra time used to detect a fault, prepare a backup, and rerun the task, these fault tolerant mechanisms may cause deadline misses if used in a real-time system.

This does not mean that passive fault tolerant structures are unsuitable for all real-time applications. It is still possible that many of the faults are tolerated within the deadlines, thus improving the system's reliability without using a more resource consuming fault tolerance mechanism. Analysis of the time use becomes necessary to determine how much the reliability can be improved while the task still meets its deadlines.

For the fault tolerant system to function properly, it is necessary to have a mechanism that determines when the active replica has failed. Using a timeout on the execution of a task is a fairly simple fault detection mechanism. If the task has not finished its execution within a given time, it is considered an omission failure. Another fault detection mechanism is to run acceptance tests [4] on results from the

task. These tests may detect some value failures, but are usually unable to determine the correctness of the results. Combining timeout and acceptance tests makes the system able to detect omission failures and some value failures.

In earlier works [6, 7], we have derived runtime models where we have focused on a fault detection mechanism where the task periodically signals that it is "alive" to the fault detector, and where it is assumed that the running replica has failed if the "alive" signals disappears. In those systems, the faults are detected a time after fault occurrence that is independent of the remaining execution time of the task. In this paper, the runtime model for systems using a combination of timeout and acceptance test for fault detection is derived.

The rest of the paper is organized as follows:

Section 2 gives a textual description of the modelled systems.

Section 3 contains the derivation of the mathematimal model for the run-time distribution.

Section 4 contains examples of the use of the model

Section 5 contains discussions and the conclusion of the paper.

2. Description of the modelled system

The class of systems that are modelled can be described as a server using passive replication mechanisms to achieve fault tolerance. In these systems, there are one active primary replica and several passive backup replicas. The replicas, as well as the fault tolerance mechanisms may be on the same node, or they may be distributed over several nodes, as shown in figure 1. The physical distribution of the system will affect the timing distributions used in the models, as well as the system's ability to tolerate some faults, but the models themselves will not be affected by the physical structure of the server.

When operating normally, the state of the primary replica is logged. The log is used when updating the passive replicas and when creating new replicas. The strategy for updating the passive replicas may vary. Frequent updates will cause more overhead during normal operation than infrequent updates, but as a result, the backups will have a state that are closer to the primary's state, so the time used to update the backups in a fault situation is shorter.

Two fault detection mechanisms are used. The timeout mechanism detects if the primary replica does not deliver results within a set maximum time, thus faults causing omission and silent failures are detected by this mechanism. The acceptance test checks the results to see if they meet criteria that all acceptable results must meet, and can thus detect some value failures.

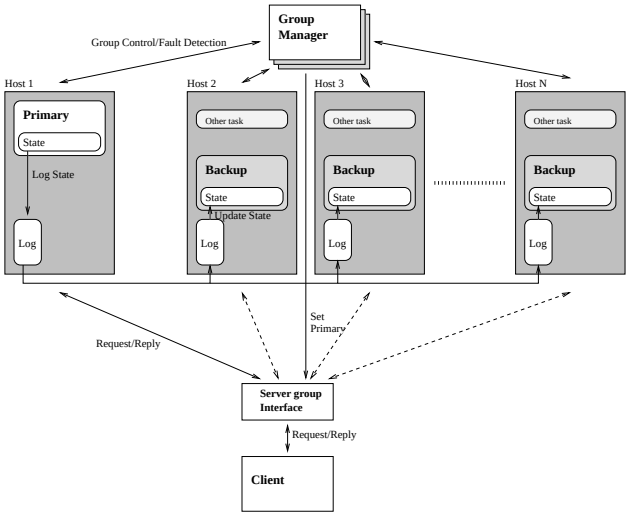

Figure 1. A passively replicated task in a distributed system

When a fault is detected, the fault tolerance mechanism prepares one of the backups to run the task. The state of the backup is updated from the log, and this backup becomes the new primary. The task is then rerun. The timing behaviour of the system when faults occur is shown in figure 2.

Faults are modeled using constant fault probabilities for each part of the execution (i.e. running the task on the primary, correction, and rerun on backup) of the system. As the faults are detected at either the timeout or after the acceptance test, we need to know that a fault has occured and whether it caused a value or an omission failure, but we do not need the exact time of the fault occurences.

There is a possibility that several faults occur during the service of one task. If a new fault occurs during the correction of previous faults or during the rerun of the task, a new detection–correction–rerun cycle will recur. If more than one fault occur before detection, the detection will be at the timeout if at least one of the faults cause an omission failure and at the acceptance test if all faults cause value failures.

3. Deriving the mathematical model

The mathematical model for the total runtime distribution of the tasks in the fault tolerant system is a function of the distributions for the fault free runtimes of the tasks, the acceptance test times, and the time between a fault detection and the rerun of the task (called correction time in this work), as well as the timeout values and the fault probabilities. The derivation of the expression is similar to the derivation of the busy period in queueing systems [2].

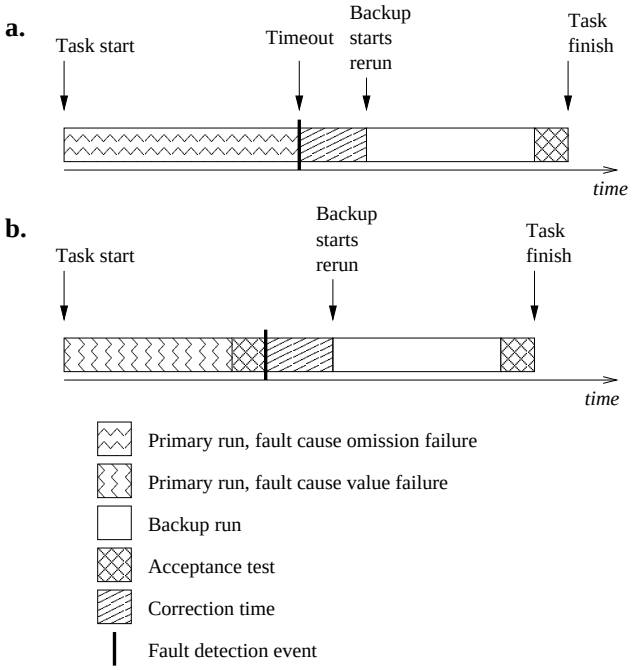

Figure 2. Timing of the fault tolerant system when a fault is detected at timeout (a) and when a fault is detected by the acceptance test (b)

The distributions used in the expressions are represented by their moment-generating functions, which can be viewed as the laplace transform of the probability density function:

$$\mathbf{F}(s) = \mathcal{L}(f(t)) = \int_0^\infty e^{-st} f(t)dt = \int_0^\infty e^{-st} dF(t) \tag{1}$$

In this work, moment-generating functions are given boldface function names (e.g. $\mathbf{F}(s)$), probability density functions are given lower-case function names (e.g. $f(t)$), and cumulative distribution functions are given upper-case function names (e.g. $F(t)$). Different represenations of the same distributions are given the same letter and index (e.g. $\mathbf{F}_\alpha(s)$, $f_\alpha(t)$, and $F_\alpha(t)$ are different representations of the same distribution).

The function names used in this work are

$\mathbf{G}(s)$**,** $g(t)$**,** $G(t)$ The runtime distribution of a task in a system where faults may occur and be tolerated, i.e. the runtime distribution that are derived in this section.

$\mathbf{M}_i(s)$**,** $m_i(t)$**,** $M_i(t)$ The fault-free runtime distribution of a task on replica i.

$\mathbf{C}_i(s)$**,** $c_i(t)$**,** $C_i(t)$ The correction time distribution, i.e. the distribution of the time used from a fault in replica $i-1$ is detected to replica i is ready to rerun the task.

$\mathbf{D}(s)$**,** $d(t)$**,** $D(t)$ The distribution of the time used for acceptance test.

$\mathbf{A}_i(s)$**,** $a_i(t)$**,** $A_i(t)$ The distribution of the time used from a failed run of the task on replica i starts to the fault detection.

3.1. The fault model

As described in the previous section, a fault model with fixed fault probability for each part of the system is used. For each replica i, there is a probability $\kappa_{\mathrm{co}i}$ that there is no omission failure while readying the replica (i.e. the correction process), there is a probability $\kappa_{\mathrm{mo}i}$ that there is no omission failure while running the replica, and there is a probability $\kappa_{\mathrm{v}i}$ that there is no value failure detected when testing the results.

If a replica fails, there are three possibilities of where the failure is detected:

- Omission failure during correction, detection is at the timeout for the correction, $\tau_{\mathrm{c}i}$
- No omission failure during correction, but omission failure during the execution of the task. Detection is at the timeout for the execution, $\tau_{\mathrm{m}i}$ after the execution started.
- No crash or omission failure, but a value failure detected by the acceptance test.

The probability for at least one of these failures occur while running replica i is given by

$$\Pr(\phi \geq 1) = 1 - \kappa_{\mathrm{co}i}\kappa_{\mathrm{mo}i}\kappa_{\mathrm{v}i} \tag{2}$$

For a run where there is no correction time (e.g. the primary replica when the task execution starts), a fault may be detected either at the timeout, $\tau_{\mathrm{m}i}$ after the execution starts, or after the normal runtime x_i and the acceptance test time z. The time to failure detection is distributed with the pdf

$$\begin{aligned} &a_{\mathrm{nc}i}(t) = \\ &\frac{1-\kappa_{\mathrm{mo}i}}{1-\kappa_{\mathrm{mo}i}\kappa_{\mathrm{v}i}}\delta(t-\tau_{\mathrm{m}i}) \\ &+\frac{1-\kappa_{\mathrm{v}i}}{1-\kappa_{\mathrm{mo}i}\kappa_{\mathrm{v}i}}\kappa_{\mathrm{mo}i}\delta(t-(x_i+z)) \end{aligned} \tag{3}$$

which has the mgf

$$\begin{aligned} &\mathbf{A}_{\mathrm{nc}i}(s) = \\ &\frac{1-\kappa_{\mathrm{mo}i}}{1-\kappa_{\mathrm{mo}i}\kappa_{\mathrm{v}i}}e^{-s\tau_{\mathrm{m}i}} \\ &+\frac{1-\kappa_{\mathrm{v}i}}{1-\kappa_{\mathrm{mo}i}\kappa_{\mathrm{v}i}}\kappa_{\mathrm{mo}i}e^{-s(x_i+z)} \end{aligned} \tag{4}$$

If there is a correction time, and a separate timeout function for this, a fault may be detected at the timeout for the

correction τ_{ci}, after the normal correction time y_i and the timeout for the runtime $\tau_{\mathrm{m}i}$, or after the correction, normal runtime and acceptance test, $y_i + x_i + z$. The time to failure detection is distributed with the pdf

$$\begin{aligned} &a_i(t) = \\ &\frac{1-\kappa_{\mathrm{co}i}}{1-\kappa_{\mathrm{co}i}\kappa_{\mathrm{mo}i}\kappa_{\mathrm{v}i}}\delta(t-\tau_{ci}) \\ &+\frac{1-\kappa_{\mathrm{mo}i}}{1-\kappa_{\mathrm{co}i}\kappa_{\mathrm{mo}i}\kappa_{\mathrm{v}i}}\kappa_{\mathrm{co}i}\delta(t-(y_i+\tau_{\mathrm{m}i})) \\ &+\frac{1-\kappa_{\mathrm{v}i}}{1-\kappa_{\mathrm{co}i}\kappa_{\mathrm{mo}i}\kappa_{\mathrm{v}i}}\kappa_{\mathrm{co}i}\kappa_{\mathrm{mo}i}\delta(t-(y_i+x_i+z)) \end{aligned} \tag{5}$$

which has the mgf

$$\begin{aligned} &\mathbf{A}_i(s) = \\ &\frac{1-\kappa_{\mathrm{co}i}}{1-\kappa_{\mathrm{co}i}\kappa_{\mathrm{mo}i}\kappa_{\mathrm{v}i}}e^{-s\tau_{ci}} \\ &+\frac{1-\kappa_{\mathrm{mo}i}}{1-\kappa_{\mathrm{co}i}\kappa_{\mathrm{mo}i}\kappa_{\mathrm{v}i}}\kappa_{\mathrm{co}i}e^{-sy_i}e^{-s\tau_{\mathrm{m}i}} \\ &+\frac{1-\kappa_{\mathrm{mo}i}}{1-\kappa_{\mathrm{co}i}\kappa_{\mathrm{mo}i}\kappa_{\mathrm{v}i}}\kappa_{\mathrm{co}i}\kappa_{\mathrm{mo}i}e^{-s(y_i+x_i+z)} \end{aligned} \tag{6}$$

3.2. Deriving the model

We start with a model of the number of detected faults, i.e. the number of reruns that is necessary to get an accepted result

$$\begin{aligned} &\Pr[\phi=k] = \\ &\begin{cases} \kappa_{\mathrm{mo}0}\kappa_{\mathrm{v}0} & ,k=0 \\ \kappa_{\mathrm{co}k}\kappa_{\mathrm{mo}k}\kappa_{\mathrm{v}k}(1-\kappa_{\mathrm{mo}0}\kappa_{\mathrm{v}0}) & \\ \quad\times\prod_{i=1}^{k-1}(1-\kappa_{\mathrm{co}i}\kappa_{\mathrm{mo}i}\kappa_{\mathrm{v}i}) & ,k>0 \end{cases} \end{aligned} \tag{7}$$

If there are k reruns, where the time to fault detection for a failed run of the task on replica i takes the time X_i, the correction time for replica k is y_k, the runtime for replica k is x_k, and the test time is z, the total runtime of the task is given by

$$Y = X_0 + X_1 + \cdots + X_{k-1} + y_k + x_k + z \tag{8}$$

We can now create an expectation function for E^{-sY} with conditions to runtimes r_n, correction times c_n and acceptance test time d and the number of faults ϕ

$$\begin{aligned} &\mathrm{E}[e^{-sY}|r_n=x_n, c_n=y_n, d=z, \phi=k] = \\ &e^{-s(X_0+X_1+\cdots+X_{k-1}+y_k+x_k+z)} \end{aligned} \tag{9}$$

As the times in this expression are independent, and by using the time to detection derived previously, i.e. $\mathrm{E}[e^{-sX_i}] = \mathbf{A}_i(s)$, the expectation function can be rewritten as

$$\begin{aligned} &\mathrm{E}[e^{-sY}|r_n=x_n, c_n=y_n, d=z, \phi=k] = \\ &\begin{cases} e^{-s(x_0+z)} & ,k=0 \\ e^{-s(x_i+y_i+z)}\mathbf{A}_{\mathrm{nc}0}(s)\prod_{i=1}^{k-1}\mathbf{A}_i(s) & ,k>0 \end{cases} \end{aligned} \tag{10}$$

Removing the condition on the number of faults is done by multiplying the probability of a given number of faults (from equation 7) with the expectation function (from equation 10) for that given number of faults, and summing the results

$$\begin{aligned} &\mathrm{E}[e^{-sY}|r_n=x_n, c_n=y_n, d=z] = \\ &\sum_0^\infty \mathrm{E}[e^{-sY}|r_n=x_n, c_n=y_n, d=z, \phi=k]\Pr(\phi=k) \end{aligned} \tag{11}$$

This yields the expectation function

$$\begin{aligned} &\mathrm{E}[e^{-sY}|r_n=x_n, c_n=y_n, d=z] = \\ &\kappa_{\mathrm{mo}0}\kappa_{\mathrm{v}0}e^{-s(x_0+z)} \\ &+\sum_{k=1}^{\infty}\kappa_{\mathrm{co}k}\kappa_{\mathrm{mo}k}\kappa_{\mathrm{v}k}e^{-s(x_k+y_k+z)} \\ &((1-\kappa_{\mathrm{mo}0})e^{-s\tau_{\mathrm{m}0}}+(1-\kappa_{\mathrm{v}0})\kappa_{\mathrm{mo}0}e^{-s(x_0+z)}) \\ &\prod_{i=1}^{k-1}(1-\kappa_{\mathrm{co}i})e^{-s\tau_{ci}}+(1-\kappa_{\mathrm{mo}i})\kappa_{\mathrm{co}i}e^{-sy_i}e^{-s\tau_{\mathrm{m}i}} \\ &+(1-\kappa_{\mathrm{v}i})\kappa_{\mathrm{co}i}\kappa_{\mathrm{mo}i}e^{-s(x_i+y_i+z)} \end{aligned} \tag{12}$$

A problem with this equation "as it is" is that there is an infinite sum in it. This can be explained as no limit to the number of faults that can occur, and thus no limit to the number of reruns that is necessary to get an acceptable result. To work around this, we set a maximum to the number of faults, N, that can occur and still be tolerated by the system. If more than N faults occur, we consider the system to have failed. Even for a relatively low N, this is a reasonable approximation of the system's behaviour, as the probability of faults occuring is usually very low. Also, because of the extra time used to tolerate faults, we can assume that a task will miss its deadline, and thus fail anyway, if more than N reruns are needed. It is also possible to model a system where there cannot be more than N faults, by setting $\kappa_{\mathrm{co}N} = \kappa_{\mathrm{mo}N} = \kappa_{\mathrm{v}N} = 1$

The conditions to time use are removed by integrating the expression with respect to the distributions, e.g., removing the condition to the acceptance test time is done with the integral

$$\begin{aligned} &\mathrm{E}[e^{-sY}|r_n=x_n, c_n=y_n] = \\ &\int_0^\infty \mathrm{E}[e^{-sY}|r_n=x_n, c_n=y_n, d=z]dD(t) \end{aligned} \tag{13}$$

Limiting the number of faults before failure to N and removing the conditions on time use for all runtimes, correction times and test times, gives the moment generating

function

$$\begin{aligned}\mathbf{G}(s) = &\\ &\kappa_{\mathrm{mo}0}\kappa_{\mathrm{v}0}\mathbf{M}_0(s)\mathbf{D}(s)\\ &+\sum_{k=1}^{N}\kappa_{\mathrm{co}k}\kappa_{\mathrm{mo}k}\kappa_{\mathrm{v}k}\mathbf{M}_k(s)\mathbf{C}_k(s)\mathbf{D}(s)\\ &\quad((1-\kappa_{\mathrm{mo}0})e^{-s\tau_{\mathrm{m}0}}+(1-\kappa_{\mathrm{v}0})\kappa_{\mathrm{mo}0}\mathbf{M}_0(s)\mathbf{D}(s))\\ &\quad\prod_{i=1}^{k-1}(1-\kappa_{\mathrm{co}i})e^{-s\tau_{\mathrm{c}i}}+(1-\kappa_{\mathrm{mo}i})\kappa_{\mathrm{co}i}\mathbf{C}_i(s)e^{-s\tau_{\mathrm{m}i}}\\ &\qquad+(1-\kappa_{\mathrm{v}i})\kappa_{\mathrm{co}i}\kappa_{\mathrm{mo}i}\mathbf{M}_i(s)\mathbf{C}_i(s)\mathbf{D}(s)\end{aligned} \tag{14}$$

Equation 14 describes the moment generating function of the runtime of a combined timeout and acceptance test system, as a function of the fault-free runtimes, the correction times, the timeouts and the probabilities that failures does not occur, and is the main result of this work.

4. Example of use

In this section, we will give some examples on how the models can be used.

For the examples a system consisting of one primary and two backups is used, i.e. equation 14 is used with $N = 2$:

$$\begin{aligned}\mathbf{G}(s) = &\\ &\kappa_{\mathrm{mo}0}\kappa_{\mathrm{v}0}\mathbf{M}_0(s)\mathbf{D}(s)\\ &+\kappa_{\mathrm{co}1}\kappa_{\mathrm{mo}1}\kappa_{\mathrm{v}1}\mathbf{M}_1(s)\mathbf{C}_1(s)\mathbf{D}(s)\\ &\quad((1-\kappa_{\mathrm{mo}0})e^{-s\tau_{\mathrm{m}0}}+(1-\kappa_{\mathrm{v}0})\kappa_{\mathrm{mo}0}\mathbf{M}_0(s)\mathbf{D}(s))\\ &+\kappa_{\mathrm{co}2}\kappa_{\mathrm{mo}2}\kappa_{\mathrm{v}2}\mathbf{M}_2(s)\mathbf{C}_2(s)\mathbf{D}(s)\\ &\quad((1-\kappa_{\mathrm{mo}0})e^{-s\tau_{\mathrm{m}0}}+(1-\kappa_{\mathrm{v}0})\kappa_{\mathrm{mo}0}\mathbf{M}_0(s)\mathbf{D}(s))\\ &\quad((1-\kappa_{\mathrm{co}1})e^{-s\tau_{\mathrm{c}1}}+(1-\kappa_{\mathrm{mo}1})\kappa_{\mathrm{co}1}\mathbf{C}_1(s)e^{-s\tau_{\mathrm{c}1}}\\ &\qquad+(1-\kappa_{\mathrm{v}1})\kappa_{\mathrm{co}1}\kappa_{\mathrm{mo}1}\mathbf{M}_1(s)\mathbf{C}_1(s)\mathbf{D}(s))\end{aligned} \tag{15}$$

For comparison, systems with the same fault mechanisms, but without one of the detection mechanisms, and where an undetected leads to the failure of the system is used.

A system without the timeout mechanisms can be modelled as a system where an omission failure will lead to an infinite response time, and can be modeled by letting the timeout values in equation 15 approach ∞ (i.e. let $e^{-s\tau} = 0$):

$$\begin{aligned}\mathbf{G}_{\mathrm{noto}}(s) = &\\ &\kappa_{\mathrm{mo}0}\kappa_{\mathrm{v}0}\mathbf{M}_0(s)\mathbf{D}(s)\\ &+\kappa_{\mathrm{co}1}\kappa_{\mathrm{mo}1}\kappa_{\mathrm{v}1}\mathbf{M}_1(s)\mathbf{C}_1(s)\mathbf{D}(s)\\ &\quad(1-\kappa_{\mathrm{v}0})\kappa_{\mathrm{mo}0}\mathbf{M}_0(s)\mathbf{D}(s)\\ &+\kappa_{\mathrm{co}2}\kappa_{\mathrm{mo}2}\kappa_{\mathrm{v}2}\mathbf{M}_2(s)\mathbf{C}_2(s)\mathbf{D}(s)\\ &\quad(1-\kappa_{\mathrm{v}0})\kappa_{\mathrm{mo}0}\mathbf{M}_0(s)\mathbf{D}(s)\\ &\quad(1-\kappa_{\mathrm{v}1})\kappa_{\mathrm{co}1}\kappa_{\mathrm{mo}1}\mathbf{M}_1(s)\mathbf{C}_1(s)\mathbf{D}(s)\end{aligned} \tag{16}$$

The timing distribution for a system without acceptance test can be derived in a way similar to equation 14, where a timing failure leads to the failure of the system, here modeled as an infinite response time:

$$\begin{aligned}\mathbf{G}(s) = &\\ &\kappa_{\mathrm{mo}0}\kappa_{\mathrm{v}0}\mathbf{M}_0(s)\\ &+\kappa_{\mathrm{co}1}\kappa_{\mathrm{mo}1}\kappa_{\mathrm{v}1}\mathbf{M}_1(s)\mathbf{C}_1(s)(1-\kappa_{\mathrm{mo}0})e^{-s\tau_{\mathrm{m}0}}\\ &+\kappa_{\mathrm{co}2}\kappa_{\mathrm{mo}2}\kappa_{\mathrm{v}2}\mathbf{M}_2(s)\mathbf{C}_2(s)(1-\kappa_{\mathrm{mo}0})e^{-s\tau_{\mathrm{m}0}}\\ &\quad(1-\kappa_{\mathrm{co}1})e^{-s\tau_{\mathrm{c}1}}+(1-\kappa_{\mathrm{mo}1})\kappa_{\mathrm{co}1}\mathbf{C}_1(s)e^{-s\tau_{\mathrm{c}1}}\end{aligned} \tag{17}$$

In the examples MATLAB and Simulink are used to numerically calculate the example systems' cumulative distribution functions from the moment-generating functions.

4.1. A basic system

In the first example, a basic replication system is modelled, to show how the expression derived in the previous chapter can be used.

The following parameters are used:

The fault-free runtimes for both the primary and the backups are distributed with a triangular distribution with minimum time 6, maximum time 10 and mode 8:

$$m_0(t) = m_1(t) = m_2(t) = \begin{cases} 0 & ,0 \le t < 6 \\ \dfrac{t-6}{4} & ,6 \le t < 8 \\ \dfrac{10-t}{4} & ,8 \le t < 10 \\ 0 & ,t \ge 10 \end{cases} \tag{18}$$

$$\mathbf{M}_0(s) = \mathbf{M}_1(s) = \mathbf{M}_2(s) = \frac{e^{-6s} - 2e^{-8s} + e^{-10s}}{4s^2} \tag{19}$$

The times used for correction is distributed uniformly between 0 and 5 for all replicas:

$$c_1(t) = c_2(t) = \begin{cases} \dfrac{1}{5} & ,0 \le t < 5 \\ 0 & ,t \ge 5 \end{cases} \tag{20}$$

$$\mathbf{C}_1(s) = \mathbf{C}_2(s) = \frac{1-e^{-5s}}{5s} \tag{21}$$

The acceptance test time is 1:

$$d(t) = \delta(t-1) \tag{22}$$

$$\mathbf{D}(s) = e^{-s} \tag{23}$$

Timeout values are set to 10 for execution of a task on a replica and 5 for the correction. The omission failure probability is set to 4×10^{-4} for execution and 2.5×10^{-4} for correction, and the value failure probability is set to 4×10^{-4}.

The resulting cumulative distribution function for this system, compared to systems with only timeout or acceptance test as a fault detection method, is shown in figure 3.

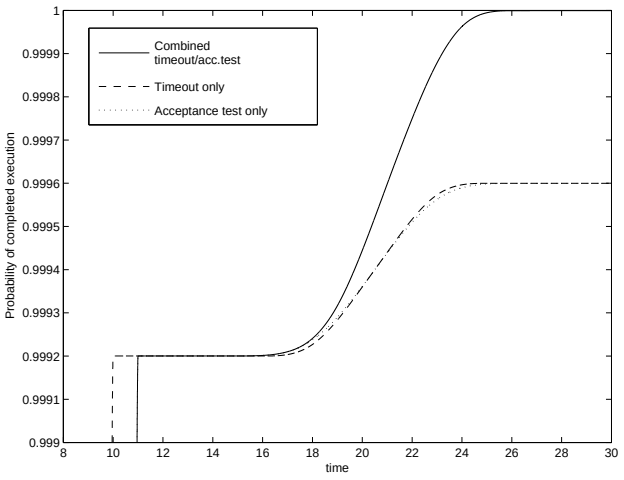

Figure 3. Cumulative distribution functions for the system described in 4.1, compared to systems with the same fault probabilities, but with only one of the fault detection methods.

The figure shows that there is very little improvement in the reliability of this system from a non-tolerant system if there is a hard deadline before $t = 20$. This is to be expected, as a task where a fault occurs must be run at least twice. Also as expected, the systems with only one of the fault detection mechanism will not achieve a failure probability lower than the probability of an undetected fault.

Between $t = 20$ and $t = 26$, the probability of completed execution increases steadily. If the combined detection mechanism is able to detect all faults except deadline faults, and there is a hard deadline, the system as a whole will only fail if all there is a missed deadline or if all the replicas fail. The probability of failure is 7.8×10^{-7} if the deadline is at $t = 26$, which is a significant improvement from the 8×10^{-4} failure probability of a non-tolerant system. For a deadline at $t = 30$, the failure probability is 5.7×10^{-7}, and for a deadline at $t = 50$ the failure probability is 8.8×10^{-10}.

4.2. A system with checkpoints

In this example, a task that can be partitioned into subtasks is investigated. Checkpoints can be placed between the subtasks, at each checkpoint, fault detection is performed, and the system's state is saved. If a fault is detected, the execution restarts from the previous checkpoint. Using checkpoints generates much overhead, and several criteria and optimization methods for the number and placement of checkpoints exists [1, 3, 8]. In this example, we will show how the derived run-time distribution model can be used to optimize the number of checkpoints using the probability of deadline miss as a criteria.

The task consists of 12 subtasks, each with a fault-free runtime that is uniformly distributed between 1 and 2:

$$\mathbf{M}_{\mathrm{sub}}(s) = \frac{e^{-s} - e^{-2s}}{s} \tag{24}$$

If a frame between to checkpoints consists of n subtasks, the fault-free runtime of the frame will be distributied as a convolution of the distribution of the n subtasks, resulting in the moment-generating function

$$\mathbf{M}(s) = \mathbf{M}_{\mathrm{sub}}(s)^n \tag{25}$$

The timeout value for a frame is set to the maximum runtime of the frame, i.e. if a frame consists of n subtasks, the timeout is set to $2n$.

The time used to run an acceptance test, in addition to any overhead related to the checkpoint is set to 1 per frame:

$$\mathbf{D}(s) = e^{-s} \tag{26}$$

The correction time is also set to 1 per frame:

$$\mathbf{C}(s) = e^{-s} \tag{27}$$

For each subtask, the probability of an omission failure during normal execution or a value failure is both set to 10^{-4}, for a frame of n subtasks, the probability of these failure types is $1 - (1 - 10^{-4})^n$. The probability of an omission failure during correction is also set to 10^{-4}.

As the execution goes back to the beginning of the frame upon fault detection, each frame can be viewed as a fault tolerant system with a timing distribution found by equation 15. If a task consists of m frames, and each frame i has the runtime distribution $\mathbf{G}_i(s)$, the total runtime distribution of the task becomes

$$\mathbf{G}(s) = \prod_{i=1}^{m} \mathbf{G}_i(s) \tag{28}$$

In this example, the task can partitioned in the following ways:

- 1 frame of 12 subtasks (i.e. no partitioning of the task)
- 2 frames of 6 subtasks
- 3 frames of 4 subtasks
- 4 frames of 3 subtasks
- 6 frames of 2 subtasks
- 12 frames of 1 subtask

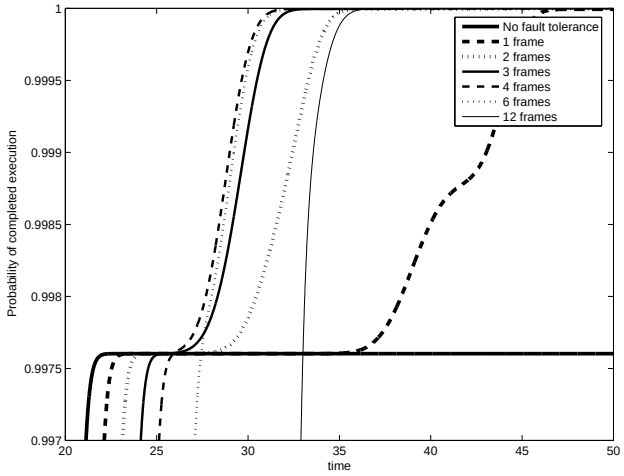

Figure 4. Cumulative distribution functions for the systems described in 4.2.

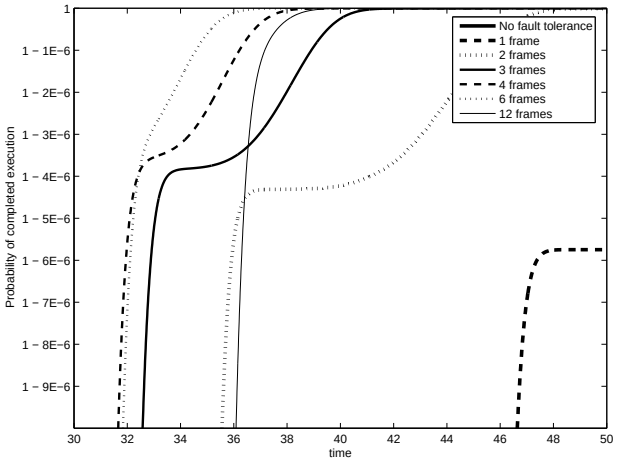

Figure 5. Cumulative distribution functions for the systems described in 4.2, details.

The resulting cumulative density functions are shown in figure 4, with closer details in figure 5.

If there is a hard deadline at t_{dl}, and the only possibilities of a system failure is a deadline miss or a single frame failing more than 2 times, the results show that a nontolerant system performs better than or as good as the other systems if $t_{dl} < 25.4$, with a failure probability of 2.4×10^3. The non-tolerant system does not have any overhead from the fault tolerance mechanisms, and there is a high probability that the other systems are not able to tolerate any fault occurences within this deadline.

For $25.4 < t_{dl} < 26.0$, the 3-frame system has a failure probability that is slightly lower than the non-tolerant system.

For $26.0 < t_{dl} < 32.4$, the 4-frame system has the lowest failure probability, and the improvement in the systems' reliability begins to show. For this system, a deadline at 30 gives a failure probability of 3.1×10^{-4}, and for $t_{dl} = 32$, the failure probability is 5.6×10^{-6}.

The 6-frame system has the lowest failure probability for $32.4 < t_{dl} < 40.9$, for $t_{dl} = 35$ the failure probability is 6.0×10^{-7}, and for $t_{dl} = 40$, the failure probability is down to 1.2×10^{-9}.

For deadilines higher than 40.9, the 12-frame system has the lowest failure probability, below 10^{-9}.

5. Discussions and conclusion

In the model presented here, a very simple fault model is used, with constant probabilities for certain failure modes. This makes the models easier to derive and use, and it is possible to approximate some other fault models, like a poisson arrival fault process, to this model. Only replica failures that are detected by the timeout and acceptance test fault detection mechanisms are considered in the model. While the timeout will cover all silent and omission failures, there is a possibility that value failures pass the acceptance tests and lead to system failures. The equations can be expanded upon to also model this behavior.

In this paper, we have derived an expression for the run-time distributions of tasks in a fault tolerant system based on passive replication, where timeouts and acceptance tests are used as fault detection mechanisms. The expression is a function of the fault-free run-time distributions of the task on the replicas, the acceptance test time distribution, the distribution of the fault correction times, and the timeout values, as well as the probabilities of non-failure of the replicas. We believe that this can be a useful tool for the analysis of fault tolerant real-time system. The use of the model has been demonstrated on example systems, showing how the model can be used to determine the probability of missed deadlines, as well as how it can be used for optimization of checkpoint placement in a hard real-time system.

References

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

[2] L. Kleinrock. *Queuing Systems Vol 1: Theory*. John Wiley and Sons, 1975.

[3] C. Krishna, K. G. Shin, and Y.-H. Lee. Optimization criteria for checkpoint placement. *Communications of the ACM*, 27(10):1008–1012, October 1984.

[4] B. Randell. System structure for software fault tolerance. *IEEE Transactions on Software Engineering*, SE-1(2), June 1975.

[5] J. A. Stankovic. Misconceptions about real-time computing: A serious problem for next-generation systems. *Computer*, October 1988.

[6] Å. Tjora and A. Skavhaug. A general mathematical model for run-time distribution in a passively replicated fault tolerant system. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, pages 295–301, Porto, July 2003.

[7] Å. Tjora and A. Skavhaug. Assessing reliability of real-time distributed systems. In *Proceedings of the 1st ERCIM Workshop on Software-Intensive Dependable Embedded Systems*, pages 59–64, Porto, August 2005.

[8] J. W. Young. A first order approximation to the optimum checkpoint interval. *Communications of the ACM*, 17(9):530–531, September 1974.