European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
A Network of Excellence funded by the European Commission

Deterministic Models of Software Aging and Optimal Rejuvenation Schedules

Artur Andrzejak
Zuse Institute Berlin (ZIB)
Takustraße 7, 14195 Berlin-Dahlem
Germany
andrzejak@zib.de

Luis Silva
Dep. Engenharia Informática
Univ. Coimbra
Portugal
luis@dei.uc.pt

CoreGRID Technical Report
Number TR-0047
November 2, 2006

Institute on System Architecture

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-004265

Deterministic Models of Software Aging and Optimal Rejuvenation Schedules

Artur Andrzejak
Zuse Institute Berlin (ZIB)
Takustraße 7, 14195 Berlin-Dahlem
Germany
andrzejak@zib.de

Luis Silva
Dep. Engenharia Informática
Univ. Coimbra
Portugal
luis@dei.uc.pt

CoreGRID TR-0047
November 2, 2006

Abstract

Automated modeling of software aging processes is a prerequisite for cost-effective usage of adaptive software rejuvenation as a self-healing technique. We consider the problem of such automated modeling in server-type applications whose performance degrades depending on the "work" done since last rejuvenation, for example the number of served requests. This type of performance degradation - caused mostly by resource depletion - is common, as we illustrate in a study of the popular Axis Soap server 1.3. In particular, we propose deterministic models for approximating the leading indicators of aging and an automated procedure for statistical testing of their correctness. We further demonstrate how to use these models for finding optimal rejuvenation schedules under utility functions. Our focus is on the important case that the utility function is the average of a performance metric (such as maximum service rate). We also consider optional SLA constraints under which the performance should never drop below a specified level. Our approach is verified by a study of the aging processes in the Axis Soap 1.3 server. The experiments show that the deterministic modeling technique is appropriate in this case, and that the optimization of rejuvenation schedules can greatly improve the average maximum service rate of an aging application.

1 Introduction

In the last decades the software applications have been highly increasing in complexity. In spite of the advances in software engineering techniques even the most mission-critical applications are still prone to some latent bugs and on top of that they are becoming more difficult to manage.

This increase in the complexity was observed, for instance, by IBM that in 2001 launched the Autonomic Computing Initiative as a vision to conduct research efforts in the area of system self-management. The topics addressed include system configuration, protection, healing and optimization. One of the four properties that is aimed is the development of self-healing computing systems. A self-healing system should be able to automatically predict or detect potential errors and to execute some proactive actions to avoid the occurrence of failures.

This paper presents a contribution to this topic of self-healing. One of the main concerns in nowadays complex software systems is the appearance of software aging. The term *software aging* describes the phenomenon of progressive degradation of the running software that may lead to system crashes or undesirable hang ups [17]. It may happen due to the exhaustion of systems resources, like memory-leaks, unreleased locks, non-terminated threads, shared-memory pool latching, storage fragmentation, data corruption and accumulation of numerical errors.

The aging phenomena is likely to be found in any type of software with enough complexity, but it is particularly troublesome in long-running applications. It is not only a problem for desktop operating systems: it has been observed

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

in telecommunication systems [1], web-servers [12, 6], enterprise clusters [5], OLTP systems [4], spacecraft systems [27]. This problem has even been reported in military systems [21] with severe consequences for the loss of lives.

There are several commercial tools that help to identify some sources of memory-leaks in the software during the development phase [26, 23]. However, not all the faults can be spotted and those tools cannot work in third-party software packages when there is no access to the source-code. This means that existing production systems have to deal with the problem of software aging.

The current state of software systems for IT systems is notably increasing in complexity with the introduction of Web-Technologies, the use of complex middleware for enterprise application integration and the usage of SOA. As a consequence, there are increasing concerns with this phenomena of software aging, and it is wise to devise some techniques to deal with this problem in order to increase dependability of autonomic capabilities of complex IT systems.

The most natural procedure to combat software aging is to apply the well-known technique of software rejuvenation [17]. Two basic rejuvenation policies have been proposed: time-based and proactive rejuvenation. Time-based rejuvenation is widely used today in some real production systems, for instance by some web-servers [12, 6]. Proactive rejuvenation has been studied in [5, 4, 13, 28, 19, 15, 18, 16, 29] and it is widely understood that this technique of rejuvenation provides better results, resulting in higher availability and lower costs.

In this paper we present some techniques that can be applied to find out the optimum rejuvenation time, with the main purpose of improving the reliability of the applications and minimizing the possible downtime due to a rejuvenation action. Our focus extends to any complex server-based software that has to run 24x7 with strict requirements of sustained performance and dependability.

In a previous experimental work with an implementation of a SOAP server Apache Axis 1.3 - we have demonstrated that this particular package of middleware is highly prone to the problem of software aging [25]. We have also found that this problem was highly deterministic which drove us our attention to develop some mathematical techniques to model the phenomena and find out the optimal strategy for rejuvenation of the computational server in single-server and cluster configurations.

This work can be applied to any software system that may potentially suffer from the problem of aging and have a deterministic behaviour: that is, even if you restart the system the problem will show up somewhere in the future, in a time-independent way, but directly related with the usage of system resources.

The aging behaviour of any software can be captured by one or more *indicators of aging.* Such an indicator is any measurable metrics of the server likely to be influenced by the software aging, for example the maximum number of requests it can serve per second. The aging indicators frequently depend on the time since last rejuvenation, but might also dependent on other metrics such as number of processed requests, number of performed database operations, size of the swap file or the amount of main memory used by the software.

The basis of this paper is a simple deterministic approach for time-independent modeling of the aging indicators. Our main assumption is that the leading indicators of performance degradation can be approximated as (deterministic) functions of some "work"-related metric, e.g. the number of served requests since last rejuvenation. This type of software degradation might be attributed to memory leaks, unterminated threads, stale file locks, and other resource depletion occurring with each request, or a series of them.

Modeling aging behaviour as a deterministic process depending on the amount of work since last rejuvenation offers several advantages. Firstly, such aging process description provides independence of the work (e.g. request) arrival rate or its time distribution. Compared to time-based characterisations these models are more universal and have less parameters. A consequence of the determinism is a simple and concise description of the model - for example, as an interpolating spline or a sequence of functions. This greatly facilitates analytical treatment and optimization - tasks much more cumbersome for probabilistic models predominant in modeling of aging phenomena [29]. Furthermore, if applicable, deterministic modeling is likely to yield a higher level of accuracy that the probabilistic techniques. Finally, the general applicability of these types of models is high since they correctly describe the aging process if memory leaks or other unreleased resourced are the primary cause of the degradation. We have confirmed this empirically for the aging processes identified in Apache Axis 1.3 (for the request rate capacity as an aging indicator).

Of course, not every aging indicator can be described as a function of the performed work. In some cases an approximate model provided by our approach might be sufficient, while in other cases deterministic models will be not applicable at all. Here a verification procedure for the applicability of the models under a given approximation error tolerance is needed. To automate the modeling process, such a verification should follow objective criteria and not rely on expert judgement. Such a model verification procedure offers an automated selection of the suitable work metric or even their combinations. We discuss in this paper the complete, automated process of choosing and fitting

the models of performance degradation with statistical testing that these models are appropriate.

The obtained models can be used to find optimal rejuvenation schedules for individual servers and server pools in order to maximize some utility function, possibly with additional constraints. For example, we might want to maximize as the utility function the average service rate capacity of a server, while ensuring that this capacity never drops below a certain limit. This technique is described and evaluated in this paper. We derive for this case analytical formulas for efficient computation of the rejuvenation times of individual servers.

The application of these techniques is illustrated and evaluated via a case study of Apache Axis 1.3 aging behaviour. Here we obtain deterministic models for the request rate capacity as a function of the number of served requests. We derive the optimal rejuvenation schedules by our technique and verify their optimality via experiments.

The main technical contributions of this paper are 1) a deterministic technique for modeling the leading indicators of aging in dependence of work metrics, 2) an automated procedure for selecting a suitable work metrics, finding a model for the aging process, and statistical verification of these results, 3) an approach for finding optimal rejuvenation schedules of applications for maximizing the average value of a performance metrics (such as the maximum service rate), 4) evaluation of our approach via a case study on the Apache Axis 1.3 server where the "work"-related metric is the number of served requests since last rejuvenation.

The rest of the paper is organized as follows: after reviewing related work in Section 2, we present the process of determining and verifying deterministic models for aging in Section 3. The next Section 4 explains how to obtain the optimal rejuvenation schedules without and with SLA constraints. Section 5 describes the setup of the experiments performed on Apache Axis 1.3 server. The evaluation of the experiments is presented in Section 6. We discuss there the obtained deterministic models for an aging indicator (maximum request rate capacity) of the server, the results of modified ANOVA tests, and the optimal rejuvenation times for maximizing the average maximum request rate capacity.

2 Related work

Software rejuvenation was first proposed in [6] and since then tens of papers have been published in the literature. Two policies have been studied to apply the software rejuvenation [21]: (a) by scheduling periodic actions for rejuvenation; (b) estimate the time for resource exhaustion and perform a technique for proactive rejuvenation.

While the first policy is simple to understand and apply it does not provide the best result in terms of availability and cost, since it may trigger unnecessary rejuvenation actions. Proactive rejuvenation is definitely a better option.

There are two basic approaches to apply proactive software rejuvenation: (i) Analytic-based approach and (ii) Measurement-based approach.

The first approach uses analytic modeling of a system, assuming some distributions for failure, workload and repair-time and tries to obtain the best optimal rejuvenation schedule. Several papers have been presented in the literature that describe analytical models. A survey about papers that follow this approach can be found in [29]. The paper presented in [17] presented a continuous-time Markov chain model to find a closed-form expression for the optimal trigger rate. In [9] was presented a semi-Markov model that relaxed the assumption for time-independent transition rates. [14] presented a Markov regenerative process that allowed the rejuvenation trigger clock to start in a robust state. A modelling approach for transactional systems was presented in [13]. This Markov-based model took into account some details of transaction arrivals and loss.

In the measurement-based approach, the goal is to collect some data from the system and then quantify and validate the effect of aging in system resources. Three main techniques have been presented in the literature: [13] used a time-based and workload-independent estimation of software aging. A different study that takes into account the workload was presented in [28]. Those authors presented then some further papers where their model has been refined. A yet different approach was used by [19] that made use of ARMA/ARX models to validate the occurrence of software aging in a web-server. The work presented in [5] considered several algorithms for prediction of resource exhaustion, mainly based on curve-fitting algorithms.

In [19] is presented a study about software aging in a Web-Server (Apache). That study uses time-series analysis to predict the occurrence of software aging. Proactive detection of software aging in OLTP servers was studied in [4] using monitoring data collected during a period of 5 months. That data was used to train a pattern-recognition tool. After the training phase, the system went back to production and kept the monitoring activity. That tool was able to predict the occurrence of software aging with a long time in advance. Another related study was presented in [15]. The authors applied MSET (a statistical pattern recognition method developed by NASA and US Department of

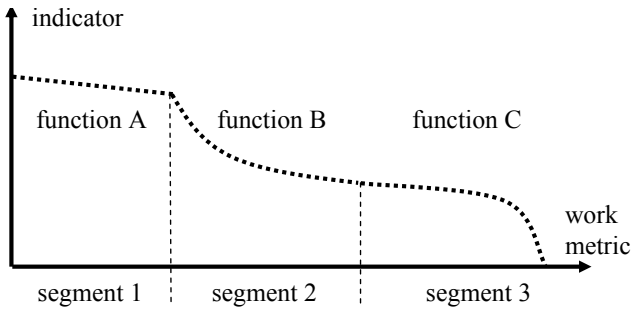


Figure 1: Approximating aging indicator by a concatenation of elementary functions

Energy) for proactive detection of software aging in cluster systems. MSET provided excellent results and was able to detect the occurrence of memory contention problems with high-sensitivity and with low probability of false-alarms. Further work was presented in [18] that kept proving the effectiveness of MSET for eager detection of software aging and runaway processes. In [16] is presented another relevant study that uses sequential probability ratio tests (SPRT) to achieve early warning of potential aging problems, by the on-line monitoring of several hardware parameters and software performance metrics.

Our modeling approach requires fitting elementary functions (polynomials, exponential functions) to noisy data from multiple trials. One of the most popular approaches to describe and approximate sampled signals is the curve fitting using splines [7]. Advanced techniques spline fitting problems in presence of noise and non-stationarity have been studied in medicine (particularly neuroscience) recently. They include DMS and SARS, and the Bayesian Adaptive Regression Splines (BARS) approach [8]. This method has been further extended to fitting curves from multiple trials [3] and to non-parametric testing of equality of functions [2]. An alternative approach for fitting splines offers approximation of data by basic, parametrized functions such as liner functions, exponential functions and higher-degree polynomials. The fitting process usually performed via the Levenberg-Marquardt algorithm [20]. The disadvantage is a difficulty of approximating the whole sample by a single function. As a remedy, a concatenation of separate functions over different argument ranges is used.

3 Modeling Aging Processes

We denote an indicator of aging by y. The amount of "work" performed by an application since last rejuvenation is described by a *work metrics* x, such as the number of served requests since reboot. The key assumption of our approach is that the indicators of aging depend primarily on a single work metrics x, i.e. $y = y(x)$. While this assumption might not apply strictly for a majority of the aging processes, it is sometimes sufficient to have approximate models of the above kind. For a user-specified approximation error level this approach can be followed if we have verification procedure to detect whether the specified error level is not likely to be exceeded. Moreover, automating such a procedure offers the opportunity to automatically test and select the work metrics (or even functions for several of them) from a pool of collected metrics. In this section we describe how to build a work metrics-based model for a given aging indicator and how to test automatically its applicability as a model input.

3.1 Approximating aging indicators by splines

Work metrics are in general monotone functions of time - they either increase or decrease. This allows for using them as arguments of other functions - in our case models for indicators of aging. Using a work metrics as an argument, we find a function $S(x)$ which approximates an aging indicator y via curve fitting. Unfortunately, an aging indicator cannot be in general described by one basic function (such as a polynomial or exponential function) over the whole range of its argument. We handle this problem by subdividing the argument range the into a set of *segments*, and fitting a basic function separately for every segment. For example, taking the maximum service rate P as an indicator of aging and the number of served requests as x, we could approximate $P = P(x)$ as a (low-order) polynomial for $x = 0, \ldots, n_1$, another polynomial for $x = n_1 + 1, \ldots, n_2$, and yet another function for $x > n_2$ (see Figure 1).

Automated and efficient procedures for fitting curves via a concatenation of low-order polynomials have been extensively studied under the term *spline fitting* [7]. A *spline* is a piecewise polynomial function $S : [a, b] \rightarrow \mathbb{R}$ consisting of k polynomial pieces $P_i : [x_i, x_{i+1}] \rightarrow \mathbb{R}$, where

$$a = x_0 < x_1 < \ldots < x_{k-1} = b.$$

The k points are called *knots*. The pieces P_i are polynomials of (commonly) degrees 3 or 4, and they specify the values of $S(t)$ over $[x_i, x_{i+1}]$, i.e. $S(x) = P_i,\ x_i \leq x < x_{i+1}$ for $i = 0, \ldots, k-2$. Their coefficients are chosen in such a way that P_i approximates in a best way the (input data) samples over $[x_i, x_{i+1}]$ and that S has a certain degree of smoothness at x_i (and x_{i+1}). The latter property is ensured by enforcing that the two pieces P_{i-1} and P_i share common derivative values from the derivative of order 0 (function value) up through the derivative of some specified order r_i. In this paper, knots correspond to the x values at which the aging indicator values have been sampled, and the intervals $[x_i, x_{i+1}]$ correspond to the segments with separately fitted basic function.

From the wide range of spline functions we are interested in *smoothing splines* [11]. Their essential property is that such splines do not necessarily pass through the original sampled points $(x_i, y_i) = (x_i, y(x_i))$. This allows more smooth curves than those strictly determined by the input data. In this way the jitter introduced by measurement errors or some secondary "noise" processes is filtered out. The degree of the spline smoothness versus the proximity to the original samples can be controlled by the *smoothing parameter* p. Formally, a smoothing spline S minimizes

$$p \sum_{i=1}^{k} \alpha_i (y_i - S(x_i))^2 - (1-p) \int \frac{d^2 S}{dx^2} dx,$$

where α_i are weights for each point (usually all 1). The smoothing parameter p is defined between 0 and 1. While $p = 0$ produces a least squares straight line fit to the data (linear regression), choosing $p = 1$ yields "perfectly fitting" cubic spline interpolant. The interesting range of p is near $1/(1 + h^3/6)$, where h is the average spacing of the data points. We assume in the following that $p = 1/(1+h^3/6)$, which allows for automated, non-parametric fitting process.

An alternative approach to spline fitting is to find the segment ends according to a visual inspection of the plot of y vs. x, and to fit a basic function from a function pool (linear, exponential, high-order polynomial) over each segment. For each segment, the function type with the best goodness-of-fit measure (such as coefficient of determination R^2 [10]) is selected. The advantage of this method is that we can have less segments, and possibly more "natural" approximating functions than piecewise polynomials. While this is mostly a manual process, it is possible to automate the selection of the segment ends via a divide-and-conquer algorithm. For each proposed segment we fit each function type and then use the goodness-of-fit measure of the "best" type as a measure for the quality of the segment selection. The quality of a choice for a collection of segments is taken as a sum of these numbers. However, the complexity of automating this procedure inclined us to prefer the spline-based method.

The process of recording the data, preprocessing and obtaining a smoothing spline approximation S consists of the following steps:

1. Select the aging indicator y and the work metric x (directly observable or a function of observables). Perform t experiments with the system under study for the same system configuration parameters (like memory size, system workload etc.). Record the samples $(x, y_i(x))$ for each experiment $i = 1, \ldots, t$. Repetition of experiments serves two purposes: filtering our of transient variations of the aging process, and allows for statistical verification of the model.

2. Optionally, generate plots of y_i as a dependent variable of x for each of the t experiments. If a visual inspection shows that the plots show different curves, select another work metrics x and return to step 1, or conclude that this modelling approach is not applicable.

3. Create an *y-average* by averaging the $y_i(x)$ values for each measurement at x (over t experiments). Thus, the y-average is a series of points $(x, y(x))$ where x is the value of the work metrics for which the measurement was taken, and $y(x) = 1/t \sum_{i=1}^{t} y_i(x)$. If the values of x are different for each experiment, they must be gridded to have same x's- in each of the t point series.

4. Use a smoothing spline algorithm described above to find a spline S approximating the y-average with the smoothing parameter $p = 1/(1 + h^3/6)$.

3.2 ANOVA-based model verification

Our assumption that an aging indicator depends primarily and deterministically on a single work metrics might not hold for more complicated aging processes. To handle this issue, we propose a test for appropriateness of our modeling technique which does not require human involvement. While the positive result of the test ensures with high confidence that our approach is applicable, a negative result does not completely exclude our modelling technique. It might just indicate that the choice of the work metrics was not correct, or possibly some function (e.g. a linear combination) of observable work metrics should be used as x. The fact that this test can be carried out automatically allows for deployment of automatic searching processes (such as genetic algorithms) for determining the most appropriate "synthetic" work metrics as a complex function of recorded system observables. When involvement of human experts is desirable, experimenting with plots of the aging indicators vs. different work metrics or their functions is an appropriate way to discover the correct work metrics.

The idea of the test is to compare the means of relative residual errors of the original sampled points $(x, y_i(x))$ versus the approximating spline S. To this aim we form t groups of the relative residuals from each of the t experiments, and additionally a group the relative residuals obtained from the spline fit of the y-average. If the model is correct, we expect the means to be all "statistically" equal to each other. The mean of the residuals from the y-average (last group) is very close to 0 by the property of the smoothing spline and the choice of the smoothing parameter p. If the model is correct, the last fact implies that all means are statistically nearly 0. In other words, in all t experiments the data shows no essential residual errors against the model. If this *null hypothesis* H_0 cannot be rejected, we can conclude with high probability that our model is appropriate. On the other hand, a test rejection of the null hypothesis gives a strong indication that at least one of the means differs, and that the model is not correct.

To verify the H_0, we use the statistical ANOVA method which analyses the variance of the relative residuals from each of the $t + 1$ groups. Essentially, the variance of the residuals over the data from all groups can be estimated by two numbers: the Mean Squared Error (MSE, or s_W) which is based on the variance within experiments, and the Mean Square Between (MSB, or s_B) which is based on the variance between experiments. If H_0 is true, then both s_W and s_B should be about the same since they are both estimates of the same quantity (total variance). However, if at least one of the means differs, MSB can be expected to be larger than MSE. Consequently, the ANOVA-test requires computation of the ANOVA F statistic:

$$F_{stat} = \frac{s_B}{s_W}.$$

Obviously, larger values of F_{stat} indicate that the null hypothesis is more likely to be wrong. To finalize the test for our F_{stat} value we need to find its p-value, i.e. the probability of obtaining an F_{stat} as large or larger than the one computed from the data while H_0 is true. If the p-value is lower that a given significance level (usually 0.05 or 0.01), the null hypothesis must be rejected, and so the model is unlikely to be correct. The p-value can be found from the F_{stat} statistics by referring to the F-distribution (sampling distribution). To use a table for this distribution, we need to specify the two degrees of freedom parameters: dfn $= K - 1$ and dfd $= N - K$, where $K = t + 1$ is the number of groups and N is the total number of samples in all groups. For further description of ANOVA see [22].

3.3 Enhancing ANOVA by a tolerance level

The above process provides an automated "yes/no" test for the appropriateness of the model for the case that the measurements from all t experiments are nearly identical. In practice this situation is rare, and so it is very helpful to have an instrument for admitting certain level of differences between the group means. Such a user-defined tolerance level allows to separate the primary model input variables from the secondary ones by neglecting the latter at the model testing stage. It also greatly facilitates in estimating the relative (mean) error of the model via a simple binary search procedure minimizing the approximation level until the value at which the model is rejected.

Unfortunately, the ANOVA method does not allow for specifying a "tolerance" value by which the data from different groups might differ but the means are still reported as equal (as most hypothesis tests, ANOVA allows for specification of the "significance level", though). As a remedy, we propose to transform the relative residuals in each group in such a way that the group means with pairwise difference of approximately 2ℓ or less become nearly equal. The parameter ℓ is a used-defined *tolerance level*. The transforming function is defined by

$$Z(r) = r\left(\frac{1 - e^{-r/\ell}}{1 + e^{-r/\ell}}\right)^2, \quad \ell > 0$$

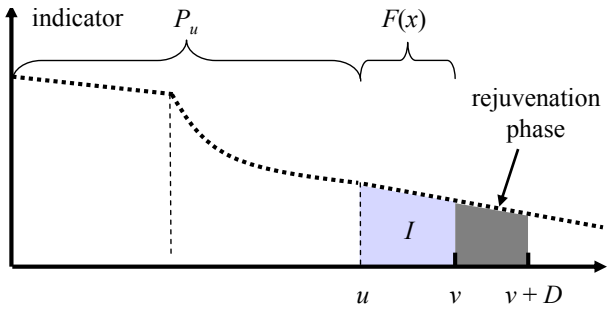


Figure 2: Role of the parameters in optimizing the average performance

and has been derived from the logistic function $1/(1 + e^{-r})$. As easily verified, this function "squashes" the values of the arguments in $[-\ell, \ell]$ to nearly 0, while mapping r to approximately $r \pm \ell$ for $|r| > \ell$. For $\ell = 0$ we set $Z(r) = r$ as the above function converges to the identity when $\ell \to 0$. After the transformation only residuals with absolute value ℓ or above are significantly contributing to the hypothesis testing.

Summarizing, an automated test of the model appropriateness consists of the following steps:

1. For each experiment i and the y-average (denoted here as y_{t+1}), compute the relative residuals $r_i(x)$ of the data versus the smoothing spline S by $r_i(x) = (y_i(x) - S(x_i))/y_i(x)$ for $i = 1, \ldots, t + 1$ and each recorded work metric value x.
2. For a given value of the tolerance level $\ell \geq 0$ transform the residuals by the function Z.
3. For a given significance level p conduct the ANOVA-test for the equality of means on the $t + 1$ data groups.

The above procedure for the model creation and testing is exemplified via a study on Apache Axis 1.3 described in Section 6.1.

4 Optimal Rejuvenation Schedules

The models of software aging presented in the previous section are important instruments for proactive software rejuvenation [28]. They can help to find automatically rejuvenation schedules which optimize some user specified utility functions or performance policies. An example for the latter is a condition that an aging indicator might never drop below a certain level L while a rejuvenation should performed as infrequently as possible. This simple yet common policy occurs typically as a part of Service Level Agreement (SLA) scenarios [25]. If a deterministic model of aging is (essentially) monotone, the solution to this problem is just trivial: the optimal rejuvenation point should be scheduled for the work metric value of $x^* - D$, where x^* is solution to the equation $L = S(x^*)$ and D is the amount of "work" (in terms of the work metric) dropped by the server during the rejuvenation phase.

A more challenging problem is the optimization of an *average* of the aging indicator over many rejuvenation cycles. Typically for this case, the aging indicator will represent some performance metric of the server. The problem here is thus to optimize the average performance while paying less attention to outage time of a server. While the latter is not desirable in the case of a single server, such scenario makes sense in a server pool. Here even frequent rejuvenation (a possible side-effect of optimized average performance) is tolerable as peer servers can handle requests of an unavailable server. In this section we treat this problem in context of our deterministic models.

4.1 Optimizing the average performance over a rejuvenation cycle

To simplify the treatment, we focus on the case that the aging indicator is the maximum number of requests which can be served by the application (the treatment also applies to other metrics). We denote this number by P and will call it *(instantaneous) performance*. We also assume that this performance depends on the number w of served requests since last rejuvenation, i.e. $P = P(w)$. Consequently, $w = x$ will be assumed to be the work metric

x in the modelling context. While it is important in SLA scenarios to keep the performance above a certain level, we are interested in averaging this number over a larger amount of work x_∞, typically including many rejuvenation cycles. This *average performance* $P_{ave}(x_\infty)$ is defined as the expression $1/x_\infty \int_0^{x_\infty} P(x)dx$ if $P(x)$ is constant over $[x, x+1]$, or as $1/x_\infty \sum_{i=0}^{x_\infty} P(i)$ otherwise. Maximizing average performance does not necessarily imply a lower bound on the instantaneous performance P (when the application is active, i.e. outside the rejuvenation phase). However, such a lower bound is interesting for SLA guarantees. Therefore, we additionally consider in the following the optimization of P_{ave} under the constraint of such a lower bound. Further discussion in this section is devoted to maximizing $P_{ave}(x_\infty)$ under different performance policies.

Recall that in our aging models $P(x)$ is approximated by a function $S(x)$ concatenated from basic functions over consecutive argument intervals (e.g. spline segments). Within each segment S is described by a different polynomial (or some basic function). The idea of our approach is to iterate over the consecutive segments, compute the optimal work metric value x^* over each one, and output the overall best solution. To simplify the notation, we identify $P(x)$ and $S(x)$ in the following.

Since we are interested in averaging performance over very large number of requests x_∞, we can assume that x_∞ is a multiple of the number of requests per rejuvenation cycle. Moreover, if the server behaves identically after each rejuvenation (and it better does), we might set x_∞ to the amount of work performed in one rejuvenation cycle. Thus, we can limit our considerations to one cycle.

For a given interval $I = [u, v]$ of the work metric value (i.e. segment of a model), the cumulative performance $P_u = \int_0^u P(x)dx$ in the previous segments might influence the solution. However, this P_u is a constant over I and can be easily computed from the information on previous segments (their boundaries and model functions) by the definition of P_{ave}. Also the start u of the interval I influences the value of $P_{ave}(x)$ for $x \in [u, v]$. Thus, while searching for the optimal rejuvenation point under the function describing $P(x)$ over I we must take P_u and u into account. Figure 2 illustrates the notation.

This leads to the following general approach. We obtain the maximum average performance $P_{ave}(x)$ by rejuvenating once the work metric has increased by x^*, with x^* determined as follows:

S1. For each segment $I = [u, v]$ of the model, assume that I contains x^*. Compute P_u and find the optimal value for $x^*(I) \in I$ as described in the next section.

S2. Compare among all segments the average performance $P_{ave}(x_\infty)$ achieved by the respective optimal solution, and output the best one $x^* = x^*(I)$ for some segment I.

4.2 Optimal rejuvenation point in a single segment

Let D be the amount of work (in terms of the work metric x) dropped by the server during the rejuvenation phase. The latter has usually a constant time, and so D is influenced by this time and the request rate distribution. This potentially introduces dependence on the request rate - something which we try to avoid. However, we might assume that the rejuvenation time is short compared to the length of the full cycle, and so the request rate during rejuvenation can be assumed as constant. Therefore, D can be estimated as a product of the rejuvenation time and the average request rate. While this is system specific, we can assume it as a constant, and so escape the dependency on the request rate distribution.

For a given segment $I = [u, v]$, the average performance P_u from $x = 0$ to $x = u$, and a given function $P(x)$ over I, we find the optimal rejuvenation point $z^* = x^* - u$ (i.e. the "offset" of x^* from the start of the segment) in the following way:

- Determine an analytical expression for the integral $F(x) = \int_0^x P(x)dx$. Here $P(x)$ is understood as the basic function over I only (transformed so that segment starts at 0).
- Find z^* which maximizes
$$P_{ave}(z) = \frac{P_u + F(z)}{u + z + D}$$
for $z \in [0, v - u]$.

The latter equation follows from the definition of $P_{ave}(x_\infty)$: the numerator sums up the cumulative performance from 0 to $u + z$, $z \in [0, v - u]$. The denominator sums up the number of requests over previous segments (u), requests over the segment start until offset (z), and the number of dropped requests during rejuvenation (D), see Figure 2.

For a basic function over I, the first step is to compute the first derivative of $P_{ave}(x)$, solve $P'_{ave}(z) = 0$ and verify that it is a maximum. If such a candidate value for z^* is within the boundaries of the current segment, we have found the optimal value and are finished. In other cases we must search for z^* numerically (e.g. via a binary search for unimodal functions) by maximizing the expression for $P_{ave}(z)$ over the current segment. In the following we discuss how to obtain z^* for the cases that $P(x)$ is linear, polynomial, or exponential function over I.

4.2.1 Linear function

For $P(x) = ax + b$ obviously $F(x) = a/2x^2 + bx$. With constants a, b, u, v and P_u, we maximize $P_{ave}(x) = (P_u + a/2x^2 + bx)/(u + x + D)$. By solving $P'_{ave}(x) = 0$ and choosing the concave downward inflection point , we find that the potential solution is

$$z* = -\frac{a(D+u) + \sqrt{a(2P_u + (D+u)(a(D+u) - 2b))}}{a}.$$

4.2.2 Polynomial function

For a polynomial of order k $P(x) = \sum_{i=0}^{k} a_i x^i$, we have $F(x) = \sum_{i=0}^{k} \frac{a_i}{i+1} x^{i+1}$. The first derivate of $P'_{ave}(x)$ is then

$$P'_{ave}(x) = \frac{\sum_{i=0}^{k} a_i x^i}{u + x + D} - \frac{P_u + \sum_{i=0}^{k} \frac{a_i}{i+1} x^{i+1}}{(u + x + D)^2}.$$

In general, the equation $P'_{ave}(x) = 0$ has a closed form for each k, but they are so complex for $k > 1$ that we suggest using a symbolic computation package [24] for finding them.

4.2.3 Exponential function

For an exponential function $P(x) = e^{ax+b}$ we have $F(x) = \frac{1}{a} e^{ax+b}$. As a potential solution we get

$$y^* = \frac{-aD - au + W\left(aP_u e^{-b+aD+au-1}\right) + 1}{a}$$

where $W(z)$ is the Lambert function, a solution t to the equation $z = te^t$ which has no closed analytical form.

4.3 Optimizing the average performance under SLA constraint

In single-server environments it is frequently desirable that the instantaneous performance $P(x)$ of a server must never drop under a certain threshold L (disregarding the rejuvenation phase). Such a condition might be imposed by an SLA or other policies, and basically guarantees that the server is never "under-performing". To solve this problem, we propose an approach similar to the one in Section 4.1, rules S1 and S2. We iterate over all segments, compute the potential solution in each case (if it exists), and select the best one among all cases. However, step S2 needs to be changed. In the new scenario we must take into account an additional constraint $P(x) \geq L$ for a $x \in [0, x^*]$ for a given segment.

In general, this is a non-linear mathematical programming problem (optimization with constraints) in one variable. Some instances of this problem might have specialized solvers. However, to reduce the solution complexity and since we are approximating the "true" aging process anyway, it is more advisable to search for an optimal and feasible solution in the following way. For each segment seg considered in S1 and its optimal solution x^*_I, we find the minimum value of P over $[0, x^*_{seg}]$. If this value is smaller than L, the solution for I is excluded from consideration. Here tolerate the error that another $x \in I$, $x \neq x^*_{seg}$ might be optimal within I under the SLA constraint, and so the segment I might contain the optimal solution. However, if the segments are fine enough (as in case of the spline-based modeling), taking a neighboring segment in this case provides a sufficient approximation.

To find the minimum value of P over $[0, x^*_{seg}]$, we iterate over all segments covering this range, and for each segment determine the minimum of its basic function by computing the extreme points (via 1st derivative) and checking the segment ends. For the last segment (containing x^*_{seg}) we apply the same process after setting its end to x^*_{seg}.

In general, finding the optimal solutions under the SLA constraints incur some computational costs, but this needs to be done only once for a given deterministic model . The latter will not change frequently if the server parameters (hardware, software settings) remain the same.

Group A	run #	1	2	3	4	5	6
	max. connections	20	20	20	25	25	25
Group B	run #	11	12	13	14	15	16
	max. connections	50	50	50	100	100	100

Table 1: Groups A and B and their parameters

5 Experimental Setup and Data Collection

We have conducted a study of dependability benchmarking with Apache Axis 1.3. The results were presented in [25]. In that study, we have used a tool of workload and stress-testing (called QUAKE) and we have used a synthetic SOAP-based web-service that resembles the behaviour of a banking application.

The testing infrastructure was composed by a cluster with 12 machines. From those 12 machines, one was running the SOAP application, other was dedicated to the Benchmark Management system; the remaining 10 machines were running instances of the clients that were in practice the workload generators. Each client node of the cluster is a machine with the following hardware characteristics: Intel celeron 1GHz with 512Mb of memory. Every node runs Linux v2.4.20 and the client modules of the QUAKE tool were implemented in Java 1.5. The SUT web-service was running on a central node (dual-processor) of the cluster. This central node had the following hardware characteristics: dual-processor AMD Opteron 64 bits 246, 2GHz, a memory of 4x1Gb (4Gb) DDR 400 and a disk SATA2 of 160Gb. In fact we have two similar central nodes with the exact same characteristics: one with Linux v2.6 that was used for some of the experiments, and an equal node with Windows 2003 Server, for some other experiments. The connection of the client nodes to the central node is done through an Ethernet switch of 100mbit/s.

The experiments were taken in a cluster with 10 machines injecting workload (doing simultaneous requests) in a central server. Each client had m threads which worked concurrently doing requests to the server. So, in the overall the maximum number of total connection was equal to $10m$.

We have recorded in the experiments the following system metrics:

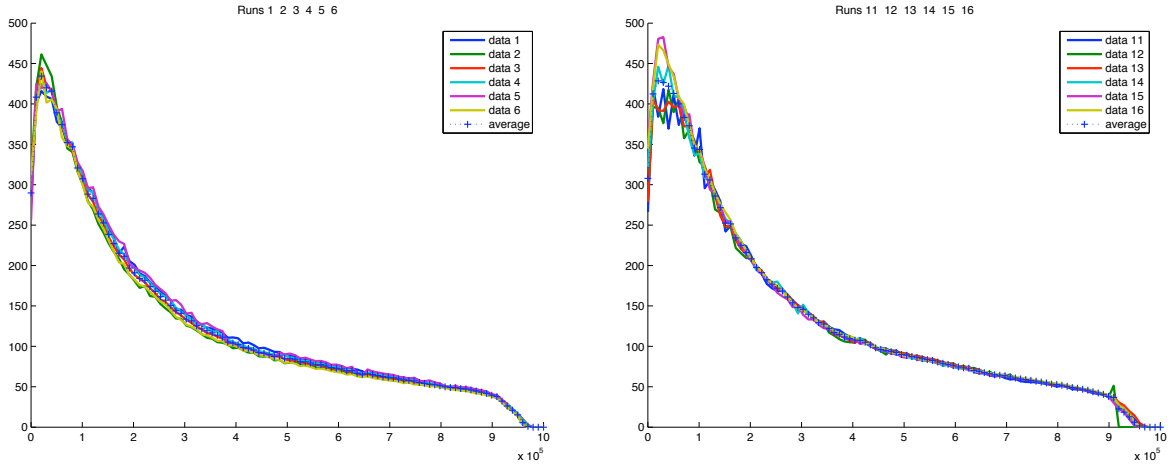
- free memory: available memory in the JVM of the SOAP server,
- cpu_user: % of CPU time used by the user applications,
- cpu_system: % of CPU time used by the operating system,
- cpu_idle: % of time where the CPU is idle,
- request_per_sec: the throughput of the SOAP server, measured in terms of number of requests than been executed per second,
- min_lat: minimum observed value for the request latency,
- max lat: maximum observed value for the request latency, and
- avg lat: average value of the request latency .

We performed 12 experiments for modeling of aging in the following way. We sent service requests to Apache Axis 1.3 with a constant rate exceeding the capacity of this server. All experiments were performed with the same parameters, except for the maximum number of total connection. We partitioned these experiments into two groups depending on these parameters. We summarize the runs and their parameters in Table 1.

It is important to note that the successor to Apache Axis 1.3 (Axis 2.0) does not suffer such severe problems of memory leaks. However, using Axis 1.3 for a study is still valid as it represents of what we believe is a common aging behaviour.

6 Empirical evaluation

The Apache Axis server can serve only a certain amount of requests per second. We have introduced this metric in Section 4.1 as the service rate capacity P or performance of a server. As this metrics is an important characteristics of



(a) group A

(b) group B

Figure 3: Server performance versus number of served requests: original (gridded) data and the y-averages

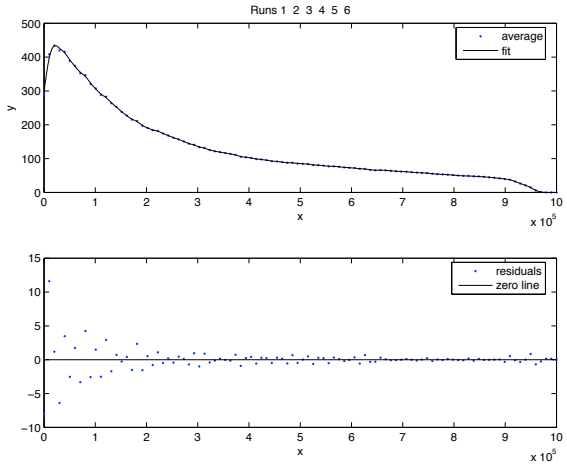
the server, we selected it as the aging indicator. The case of a server-type application offers a very natural work metric: the number of requests served w since last rejuvenation. We have tested all the recorded metrics (except number of requests per second) as potential work metrics, but the number of requests served turned out to be the best choice. We assume the choices $x = w$ and $y = P$ in the following.

6.1 Modeling service rate capacity of Apache Axis 1.3

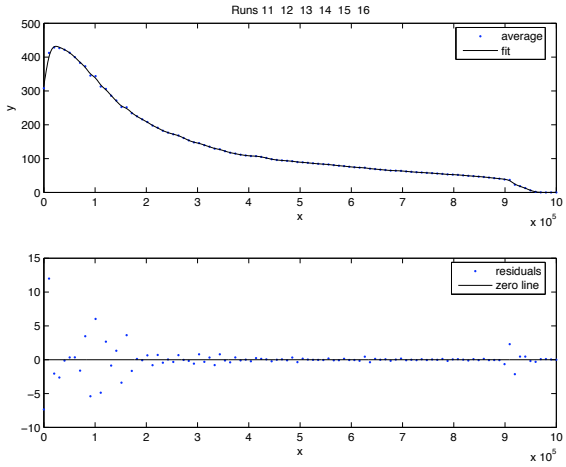
In this section we demonstrate the modeling approach to the aging process of the Apache Axis 1.3 server. For each of the groups, we have first performed the gridding of the data to obtain equal values of the work metrics in each run. To this aim, we have subdivided the recorded range $[0 \dots 10^6]$ of w into 100 equally-sized bins, and took the bin start as the gridded w value. The gridded P value has been obtained by averaging the P values of the samples falling into the bin.

According to the modeling procedure from Section 3.1, we computed the y-average by taking the mean of all 6 P values belonging to the same gridded w value. The smoothing splines were fitted over this y-average using the default smoothing parameter $p = 1/(1 + h^3/6) = 6 \times 10^{-12}$ (with $h = 10^4$). Figure 3 shows the gridded data along with the y-averages, while Figure 4 presents the fits of the smoothing splines along with the absolute residuals of the y-averages against the fitted splines. We also include the plots the relative residuals of the original gridded data against the splines in Figure 5 (the vertical axis has been capped to interval $[-10\%, 10\%]$ which excluded some outliers). These plots reveal some systematic errors: in group A, runs 2 and 6 have largest of them, while in group B the largest systematic error occurs for run 15. However, almost all of the relative errors remain in the interval $[-10\%, 10\%]$ which is seemingly the influence range of secondary, not modeled factors and transient phenomena.

Figure 6 shows the box and whisker plots of the Z-transformed relative residuals for tolerance level $\ell = 5\%$ (few outliers outside $[-10\%, 10\%]$ are not shown). The boxes have lines at the lower quartile, median, and upper quartile values. The whiskers extend to the most extreme data value within 1.5 times the interquartile range of the sample. Obviously the relative residuals for group B have less differences between the means (and medians) than those from group A. The application of the Z-transformation has moved the residuals closer together (and to 0) as in the non-transformed data as intended (cf. Figure 5). The results of the ANOVA-analysis for different tolerance levels ℓ are shown in Table 2. We have highlighted the p-values at tolerance levels for which the model can be accepted at the significance level $p = 0.05$. A deterministic model for group A can be accepted at tolerance $\ell = 6\%$ (or higher), while for group B this is already the case for $\ell = 1\%$. However, in the latter case increased tolerance level up to $\ell \leq 10\%$ do not translate to higher assurance that a model is correct. We attribute this to the fact that most of the between group variance (or equivalently, MSB) comes from the outliers outside the tested tolerance levels, which are not affected by

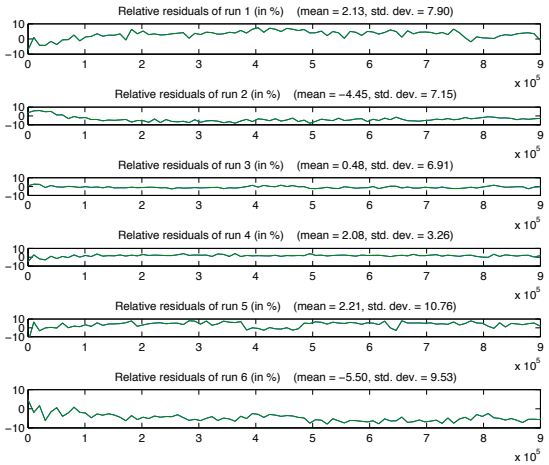


(a) group A

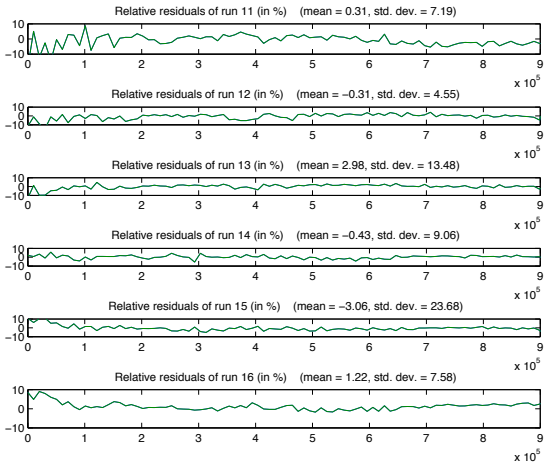


(b) group B

Figure 4: Spline fits and the absolute residuals of y-averages

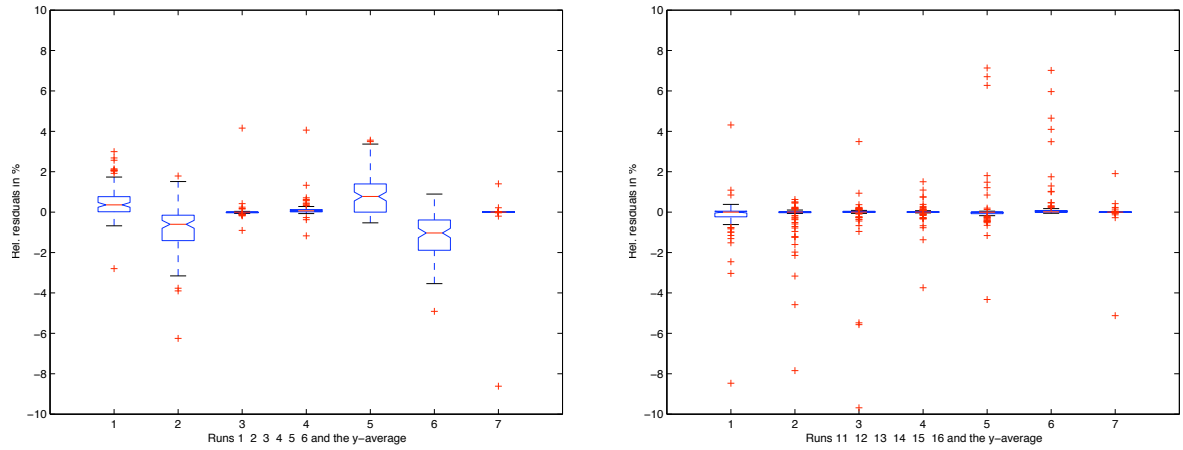


(a) group A



(b) group B

Figure 5: Relative residuals of the original (gridded) data against the spline fit (outliers outside $[-10\%, 10\%]$ are not shown)



(a) group A

(b) group B

Figure 6: Bar plots of the relative residuals of the original (gridded) data against the spline fit for $\ell = 5$ (outliers outside $[-10\%, 10\%]$ are not shown)

	ℓ	0	1	2	4	6	7	8	9	10	15	20
A	F_{stat}	18.86	16.84	10.49	3.73	2.09	1.77	1.57	1.44	1.34	1.08	0.94
	p-value	0	0	0	0.001	**0.053**	**0.103**	**0.152**	**0.197**	**0.236**	**0.375**	**0.463**
B	F_{stat}	2.40	2.10	1.91	1.90	1.98	2.00	2.02	2.02	2.01	1.90	1.75
	p-value	0.027	**0.051**	**0.077**	**0.078**	**0.067**	**0.063**	**0.062**	**0.061**	**0.062**	**0.078**	**0.108**

Table 2: Results of the ANOVA-tests for different levels of the tolerance ℓ

the transformation Z. Summarizing, we conclude that for both groups the spline functions of the number of served requests w can be accepted as deterministic models for the aging process with reasonable tolerance level $\ell = 6\%$.

6.2 Optimizing rejuvenation times

Group	D	0	500	1500	3000	4500
A	x^*	43812	45925	49700	55091	60156
	$P_{ave}(x^*)$	405.87	401.39	393.25	382.59	373.35
Group	D	0	500	1500	3000	4500
B	x^*	51862	55014	60146	66312	71760
	$P_{ave}(x^*)$	411.10	407.30	400.41	391.34	383.36

Table 3: Optimal rejuvenation points x^* and the corresponding average performance $P_{ave}(x^*)$

We used the spline models obtained in the previous section to determine the rejuvenation schedules optimizing the average performance P_{ave}. We applied in the optimization process from Section 4.1 the formulas stated in Section 4.2.2 as they apply to polynomials of degree 3 provided by the spline models. The optimizations have been carried out for different values of D (number of requests dropped during the rejuvenation) and different SLA levels L. As the maximum rejuvenation time of the Apache Axis 1.3 server was 10 seconds and its peak performance was below 450 requests per second, the maximum number of requests during the rejuvenation is 4500. To incorporate different (constant) request rates of 0, 150, 300 and 450 we studied D values of 0, 1500, 3000 and 4500. The parameter L has been studied from 0 (no SLA condition) up to 300, which was approximately the initial performance of the Axis 1.3 server.

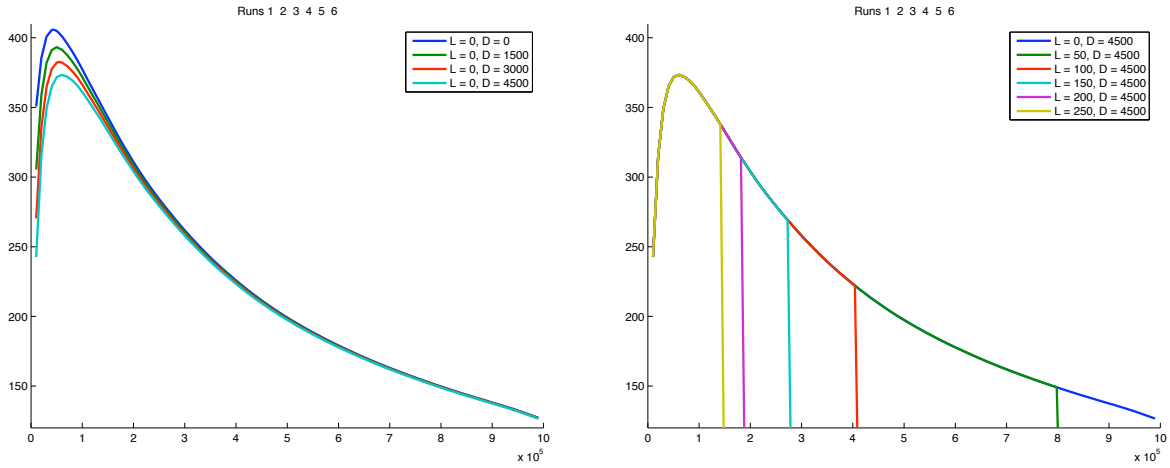


Figure 7: Average performance P_{ave} plots for group A depending on the rejuvenation point: varied numbers of dropped requests during rejuvenation D (left, for $L = 0$) and varied SLA levels L (right, for $D = 4500$)

Group	L	50	100	200	250	300
A	α_L	797980	404040	181820	141410	0
	$P_{ave}(\alpha_L)$	149.11	222.12	314.16	337.98	0
Group	L	50	100	200	250	300
B	α_L	818180	434340	202020	15152	121210
	$P_{ave}(\alpha_L)$	155	226.63	321.21	350.31	367.55

Table 4: Maximum rejuvenation points before SLA violation α_L and the corresponding average performance $P_{ave}(\alpha_L)$

Figure 7 shows the plots of the average performance P_{ave} depending on the rejuvenation point for group A (group B produced similar results). The peaks of these curves determine the optimal rejuvenation points. In the left figure, we varied D. With increased values the optimal rejuvenation points came later (after larger number of served requests) and the maximum average performance decreased as expected. Table 3 shows the exact values of the optimal rejuvenation points and the corresponding average performance for both groups A and B. The optimal rejuvenation points are quite early in the whole cycle. Interesting are the relatively sharp peaks of the curves in Figure 7. This underlines the importance of the optimizing the rejuvenation times to attain a high level of average performance, since even small deviations from the peak result in large drops of P_{ave}.

In the right figure we fixed D to 4500 and regarded different SLA levels L. The drop or *cut-off* α_L of each curve to 0 (not plotted) occurs at a x-value for which the SLA cannot be fulfilled any more. In other words, only if the rejuvenation point is in interval $[0, \alpha_L)$, the performance P of the server never drops below L. The values of cut-off points are valid also in a scenario when we only want to observe the SLA condition, without maximizing the average performance. Table 4 shows the cut-off points and corresponding values of the average performance for different values of L. In the case of the Axis 1.3 models, the optimal rejuvenation points are always before the cut-off points, and so the same as without the SLA condition.

6.3 Modeling error sensitivity analysis

The introduction of the tolerance level ℓ raises the question about the sensitivity of the optimal rejuvenation points to the errors in the data used for modeling of the aging process. To investigate this relation, we have distorted the y-average data by multiplying each value with a random error $E = 1 + r$, where r was a random number drawn from a normal distribution with mean 0 and standard deviation of $d/100$. We call d as the *distortion factor*. Thus, each point

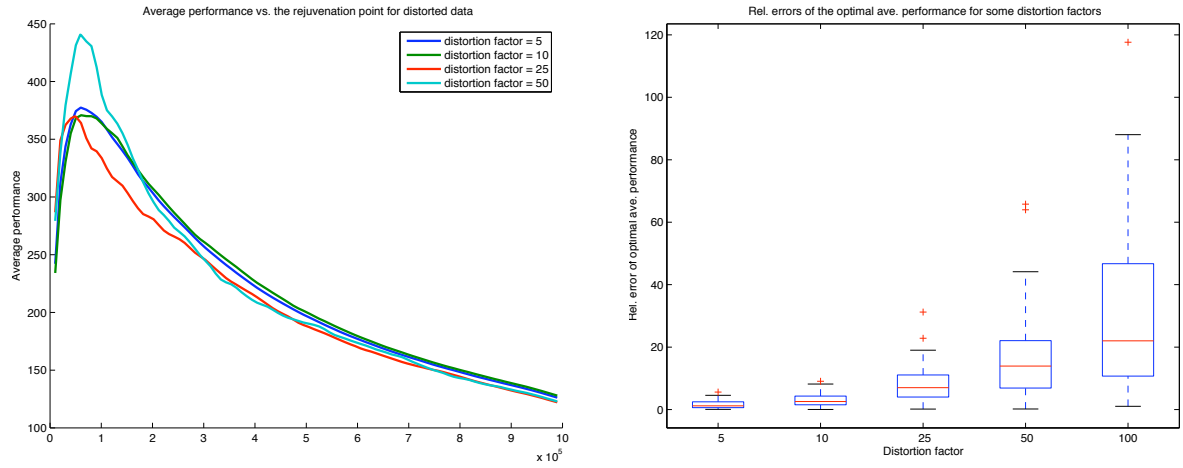


Figure 8: Average performance P_{ave} plots for distorted data (left); relative absolute errors of the maximized average performance for different distortion factors (right) ($D = 4500$, $L = 0$)

of the original data has a high probability to be distorted by d percent of its value, and lower probability for higher deviations. The value of the distortion factor relates to the value of the tolerance level ℓ: data which deviates from the y-average by a distortion factor of $d \leq \ell$ will not cause the model rejection for a given value of ℓ.

The plots of the average performance vs. the rejuvenation points for different distortion factor values are shown in the left figure of Figure 8 (group A, $D = 4500$ and $L = 0$). In the right figure of Figure 8 we show the relative absolute errors of the average performance P_{ave} for the optimal rejuvenation points computed for the distorted data. In other words, after computing the maximized average performance $v_1 = P_{ave}(x^*)$ for the original data (i.e. the y-average), we have computed the analogous number v_2 for distorted data, and calculated the relative absolute errors $\text{abs}(v_1 - v_2)/v_1$. We repeated this procedure 50 times for each of the distortion factors. The meaning of the box and whisker lines is the same as in Section 6.1. Obviously the relative errors of the maximized average performance are sub-linear in the distortion factor d, which shows some robustness of $P_{ave}(x^*)$, i.e. small sensitivity to errors.

7 Conclusion

Self-management is a topic of interest not only for the academia but also for the IT Industry that has now in hands a problem of unprecedented complexity in the software-based systems, that is increasing the total cost of ownership and the impact of dealing with failures. In this paper, we have presented a contribution in the domain of self-healing, proposing an automated approach for deploying adaptive software rejuvenation.

Our method is based on finding deterministic, request rate independent models of aging processes and a statistical test for verifying their correctness. On top of these models we proposed an approach for finding optimal rejuvenation points under certain utility functions related to average server performance. These software rejuvenation techniques can be generalized to any software application that may present some deterministic pattern of software aging. The experimental evaluation of our technique using the Apache Axis v1.3 has illustrated that accurate aging models can be achieved by the proposed methods, and showed the importance of optimizing the rejuvenation points for maximizing the average server performance.

Future work in this domain will focus on two topics: improvement of model adaptivity and the generalization of the optimization technique to server pools. For the former topic, we plan to incorporate into the model the running values of application metrics (e.g. aging indicators) to achieve higher adaptivity via *on-line* corrections of the model. Concerning the server pools, we want to find optimal rejuvenation schedules for pools of servers with heterogeneous aging behaviour.

8 Acknowledgements

This research work is carried out in part under the FP6 Network of Excellence Core-GRID funded by the European Commission (Contract IST-2002-004265).

References

[1] A. Avritzer and E. Weyuker. Monitoring smothly degrading systems for increased dependability. *Empirical Software Engineering*, 2(1):59–77, 1997.

[2] S. Behseta and R.E. Kass. Testing equality of two functions using bars. *Statistics in Medicine*, 24:3523–3534, 2005.

[3] S. Behseta, R.E. Kass, and G. Wallstrom. Hierarchical models for assessing variability among functions. *Biometrika*, 92:419–434, 2005.

[4] K. Cassidy, K. Gross, and M. Malckpour. Adavanced pattern recognition for detection of complex software aging phenomena in online transaction processing servers. In *Proceedings of the 2002 Int. Conf. on Dependable Systems and Networks DSN-2002*, 2002.

[5] V. Castelli, R. Harper, P. Heidelberg, S. Hunter, K. Trivedi, K. Vaidyanathan, and W. Zeggert. Proactive management of software aging. *IBM Jorunal Research & Development*, 45(2), March 2001.

[6] Microsoft Corporation. Technical overview of internet information services (iis) 6.0.

[7] Davis. B-splines and geometric design. *SIAM News*, 29(5), 1997.

[8] I. DiMatteo, C. R. Genovese, and R. E. Kass. Bayesian curve-fitting with free-knot splines. *Biometrika*, 88:1055–1071, 2001.

[9] Tadashi Dohi, Katerina Goseva-Popstojanova, and Kishor S. Trivedi. Statistical non-parametric algorithms to estimate the optimal software rejuvenation schedule. In *Pacific Rim International Symp. Dependable Computing (PRDC 2000)*, pages 77–84. IEEE Computer Soc. Press, December 2000.

[10] N. R. Draper and H. Smith. *Applied Regression Analysis*. John Wiley & Sons, New York, 3 edition, 1998.

[11] R.L. Eubank. *Spline Smoothing and Non-parametric Regression*. M. Dekker, New York, 1988.

[12] Apache Foundation. Apache performance tuning.

[13] S. Garg, A. van Moorsel, K. Vaidyanathan, and K. Trivedi. A methodology for detection and estimation of software aging. In *Proceedings of the 9th Int'l Symposium on Software Reliability Engineering*, pages 282–292, 1998.

[14] Sachin Garg, Antonio Puliafito, Miklos Telek, and Kishor S. Trivedi. Analysis of preventive maintenance in transactions based software systems. *IEEE Transactions on Computers*, 47(1):96–107, 1998.

[15] K. Gross, V. Bhardwai, and R. Bickford. Proactive detection of software aging mechanisms in performance critical computers. In *Proceedings of 27th Anual IEEE/NASA Software Engineering Symposium*, December 2002.

[16] K. Gross and W. Lu. Early detection of signal and process anomalies in enterprise comuting systems. In *Proceedings of the 2002 IEEE International Conference on Machine Learning and Applications ICMLA*, June 2002.

[17] Y. Huang, C. Kintala, N. Kolettis, and N. Fulton. Software rejuvenation: Analysis, module and applications. In *Proceedings of Fault-Tolerant Computing Symposium FTCS-25*, June 1995.

[18] K. Kaidyanathan and K. Gross. Proactive detection of software anomalies through mset. In *Workshop on Predictive Software Models (PSM)*, September 2004.

[19] L. Li, K. Vaidyanathan, and K. Trivedi. An approach for estimation of software aging in a web-server. In *Proceedings of the 2002 International Symposium on Empirical Software Engineering (ISESE'02)*, 2002.

[20] Donald Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM J. Appl. Math.*, 11:431–441, November 1963.

[21] E. Marshal. Fatal error: How patriot overlooked a scud. *Science*, page 1347, March 1992.

[22] R. G. Miller. *Beyond ANOVA: Basics of Applied Statistics*. Chapman & Hall, Boca Raton, FL, 1997.

[23] Parasoft. Parasoft homepage.

[24] Wolfram Research. *Mathematica*. Wolfram Research Inc., Champaign, Illinois, 5 edition, 2005.

[25] L. Silva, H. Madeira, and Silva J. G. Software aging and rejuvenation in a soap-based server. *IEEE-NCA: Network Computer and Applications*, July 2006.

[26] Scitech Software. Memprofiler – .net memory profiler.

[27] A. Tai, S. Chau, L. Alkalaj, and H. Hect. On-board preventive maintenance: Analysis of effectiveness and optimal duty period. In *Proceedings Third International Workshop on Object-Oriented Real-Time Dependable Systems*, February 1997.

[28] K. Vaidyanathan and K. S. Trivedi. A measurement-based model for estimation of resource exhaustion in operational software systems. In *Proceedings of 10th IEEE Int'l Symposium on Software Reliability Engineering*, pages 84–93, November 1999.

[29] K. Vaidyanathan and K. S. Trivedi. A comprehensive model for software rejuvenation. *IEEE Trans. Dependanble and Secure Computing*, 2(2):1–14, April-June 2005.