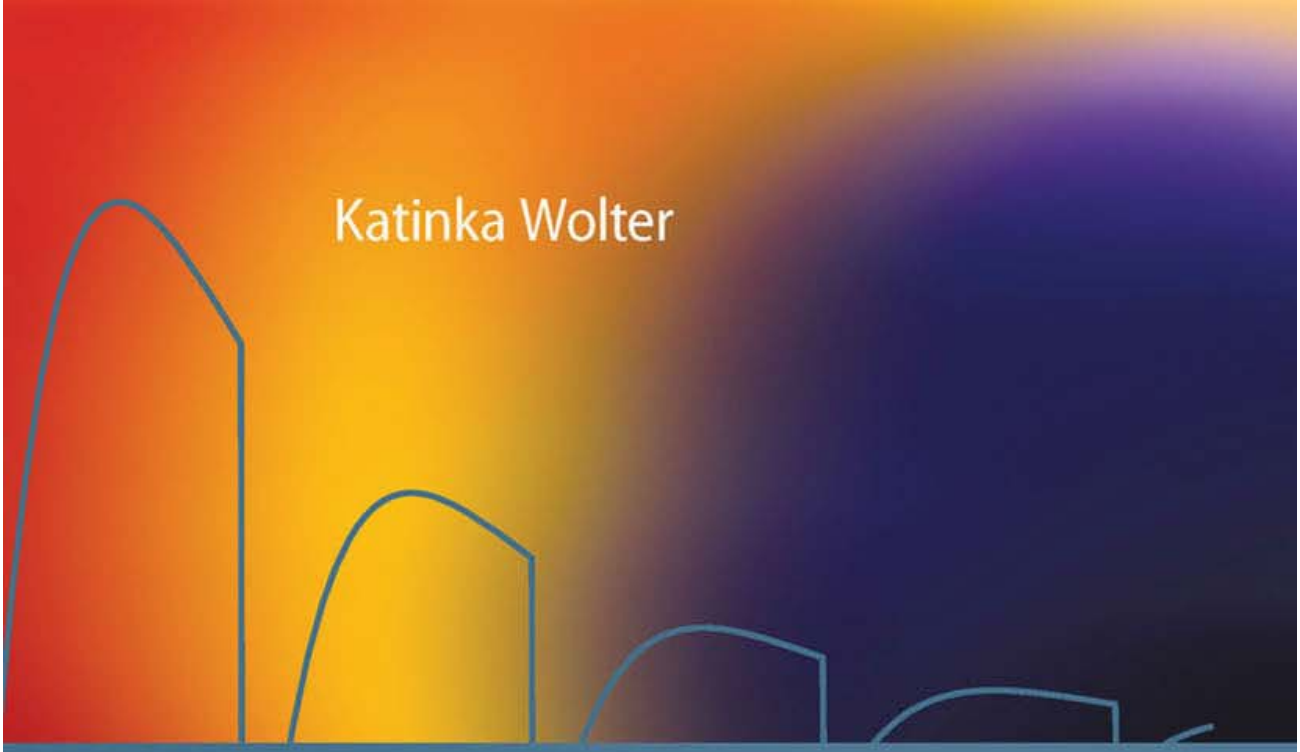

Katinka Wolter

Stochastic Models for Fault Tolerance

Restart, Rejuvenation and Checkpointing

Springer

Stochastic Models for Fault Tolerance

Katinka Wolter

Stochastic Models for Fault Tolerance

Restart, Rejuvenation and Checkpointing

Springer

Katinka Wolter
Institute of Computer Science
Freie Universität Berlin
Takustr. 9
D-14195 Berlin
Germany
katinka.wolter@fu-berlin.de

ISBN 978-3-642-11256-0 e-ISBN 978-3-642-11257-7
DOI 10.1007/978-3-642-11257-7
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2010921028

ACM Computing Classification (1998): C.4., G.3, D.4.8

Cover design: KuenkelLopka GmbH, Heidelberg

Printed on acid-free paper.

Springer is part of Springer Science+Business Media (www.springer.com)

To Tobias and our children, Louise,
Margarete, Elisabeth, Samuel and Jakob

Foreword

The book you hold in your hands, entitled Stochastic Models for Fault Tolerance - Restart, Rejuvenation and Checkpointing, found its incarnation in the hidden wonders of the Internet. On the face of it, the Internet may look like just any other network, such as the telephony network of old, cable TV network, satellite communication networks or bank transfer networks. However, when considering the actual usage patterns of the Internet, an exciting world of mathematical interest and curiosity opens up, to which that of other networks pale in comparison. Caused by the sheer number of users and web services, as well as the elegant intricacy of the packet-based network technology, measurements of the Internet have revealed highly intriguing patterns. These patterns exhibit such phenomena as the small world effect, scale free networks and self-similarity, each of which one can find discussed extensively in popular and scientific literature alike.

The work reported in this book came about because of another interesting Internet phenomenon, namely that of the 'heavy tail'. It says that high download times are relatively common, much more common than with the thin exponential tail that characterizes completion times in traditional communication networks. This has a fascinating consequence, which under certain assumptions can be proven mathematically: it is often faster to abort and retry a download attempt than to wait for it to complete. After all, one may have been caught in the heavy tail, from which one can only escape by clicking the reload button. This fascinating fact seduced us into conducting research into the optimal timing of these retries (or restarts as they will be called in this book), eventually leading to Katinka Wolter's Habilitation and this book.

Our research resulted in a number of interesting theoretical results, accompanied by extensive experimental work (mostly carried out by Philipp Reinecke). In this book, Katinka Wolter embeds these results into an extensive survey of existing scientific literature in restarts, rejuvenation, checkpointing and preventive maintenance in general. These categories of techniques have in common the problem of timing: how frequent should one carry out the preventive or pre-emptive activity? Mathematically, this leads to a set of related problems and solutions, and this book provides the reader with a careful overview of the various mathematical results as well as their inter-relations.

Many authors have researched problems of timing with great success, and the book highlights the various contributions. From a historical perspective, computing related research on timing was preceded by that on preventive maintenance of systems in general, such as for engine parts, machinery, and so forth. The results for computing systems build on these generic insights, roughly in three phases. First, mostly in the eighties of the past century, the question of when to save intermediate results (a 'checkpoint') became important. If a long-running computation fails, a checkpoint assures one does not need to start from scratch, but can roll back to the last checkpoint. The mathematical problem then is to compute the checkpoint interval that minimizes the completion time of the computation, under various fault and load assumptions.

In the nineties, researchers became interested in rejuvenation, especially when it was proposed as a generic fault tolerance approach by Bell Labs researchers. In rejuvenation, a computer will be switched off and on deliberately to avoid imminent more serious uncontrolled crashes. The question is when to 'rejuvenate' the computer. This can be based solely on a timer, but as the survey in this book shows, rejuvenation models tend to become more complex since the key for this problem is to identify the state in which to rejuvenate the system. Finally, retry problems, such as in Katinka Wolter's main line of research, have been discussed throughout the history of computing. In some sense, retries are the dual of preventive maintenance: both deal with the life time of a process, but maintenance tries to prolong life (of a system), while retries try to shorten life (of a job) as fast as possible. This fundamental difference introduces subtle differences in the mathematical models and makes that problems of interest in preventive maintenance do not always translate to retries. In the respective chapters, this book will illustrate these differences.

Readers of this book will find an up-to-date overview of the key theoretical results for timing restarts, rejuvenation and checkpointing, thus serving as a hand book for engineers facing this issue. Computer researchers may find inspiration in the elegance of many of the results, and contribute their own extensions or improvements. Non-specialist readers may hopefully catch a glimpse of the wonders hidden inside the Internet, which emerge from the combined behaviour of all of us browsing the web, downloading music and emailing friends. For all readers, the next time you are browsing the web and a page takes long to download, you may think of this book and use it to time your clicking of the reload button.

It has been a sincere pleasure to have been associated with the research that eventually accumulated into this book. It was great to work with Katinka as well as Philipp solving some of the research problems. I congratulate Katinka Wolter with completing the challenging but extremely valuable task of writing this book and trust the result will be solid information and occasional enjoyment for many readers.

Newcastle, UK, Aad van Moorsel
October 2009

Preface

As modern society relies on the fault-free operation of complex computing systems, system fault-tolerance has become a matter of course. Common agreement exists that large software systems always contain faults and precautions must be taken to avoid system failure. Failure of hardware components often is caused by external factors that can be neither predicted, avoided, nor corrected. Therefore, mechanisms are needed that guarantee correct service in the presence of failure of system components, be it software or hardware elements. Commonly used are redundancy patterns. These can be either redundancy in space or redundancy in time. Redundancy in space means the inclusion of additional hardware or software modules in the system that can replace a failed component. Different types of redundancy in space exist, such as cold, warm and hot stand-by. Redundancy in time includes methods such as restart, rejuvenation and checkpointing, where execution of tasks is repeated, or the task environment is restarted, either following a system failure or, preventively, during normal operation.

This book is concerned with methods of redundancy in time that need to be issued at the right moment. In particular we address the question of choosing the right time for the different fault-tolerance mechanisms. This includes a brief introduction to the regarded methods, i.e., restart, rejuvenation and checkpointing and aspects of their practical implementation in real-world systems. But the focus of the book is on selecting the right time, or timeout, for restart, rejuvenation and checkpointing. In general, this is the timeout selection problem.

Selecting the right timeout is a problem that is subject to a number of uncertainties. It is, in general, unknown when the system will fail. Furthermore, upcoming busy and idle periods of the system are not known, as is future user interaction with the system. Due to the many uncertainties, the timeout selection problem lends itself for a stochastic treatment. Consequently, many stochastic models addressing the timeout selection problem in restart, rejuvenation and checkpointing have been proposed. This book gives an overview of existing stochastic models of restart, rejuvenation and checkpointing.

The second part of this book treats a stochastic model of restart in various facets. Restart operates on program, or application level. If a task does not complete within a given time it is suspected to have failed and is, consequently, aborted and restarted. The timeout after which to abort and restart the task must be carefully chosen

because if it is too short the task might be aborted just before completion while if it is chosen too long one must wait unnecessarily.

The second method is software rejuvenation. Software rejuvenation restarts the operating environment of a task in order to prevent failures. Rejuvenation is a proactive fault-tolerance treatment. Hence it is issued before the system fails. This is particularly challenging, as it implies assumptions as to when the system would fail if no measures were taken. Ideally, the rejuvenation interval would always end just before the system fails. A conservative choice of the rejuvenation interval will select short intervals. But rejuvenation comes with a cost of saving the operating environment and all processes, restarting the system and reinitialising the operating environment and all processes. If rejuvenation is performed too often the rejuvenation cost accumulates unnecessarily, while if the system is rejuvenated at too long intervals it will often fail, rendering rejuvenation uneffective.

The third method, checkpointing, is the most complex mechanism of the three as it has a preventive component, saving a checkpoint, and a reactive component, rollback recovery. Checkpointing systems save the system state in regular or irregular time intervals. Upon failure the system recovers by rolling back to the most recent checkpoint. The work performed since the most recent checkpoint is lost with a failure. If checkpoints are taken too frequently the interrupt and save operation incurs too high a cost, while if checkpoint intervals are too long much work is lost upon system failure.

For all three methods similar trade-offs exist. The fault-tolerance mechanisms come at a cost that must be traded against the cost of a potential system failure. If the timeouts are well chosen the fault-tolerance mechanism will avoid a failure and be worth-while. The trade-offs can be evaluated and optimised using stochastic models. The focus of this book is to collect, summarise and compare those stochastic models. This can be seen as a first step towards understanding and solving the generic timeout selection problem.

This book is based on the author's habilitation thesis at Humboldt-University in 2008. The habilitation thesis, and hence this book, would not be as it is without the careful and thorough reading from the first to the last page of Mirek Malek. I would like to thank him for his efforts. His many valuable comments helped to improve this text tremendeously. I am thankful to Boudewijn Haverkort and Miklos Telek as they agreed on reviewing and commenting on the thesis. Miklos Telek even came to Berlin for the habilitation lecture, even though it was on yet another topic.

I thank the members of the habilitation committee for their invested time and consideration. It was a pleasure to work with the gifted students at Humboldt-University over the five years I spent there until completion of my habilitation thesis. I particularly thank Willi Engel for being a member of the habilitation committee and Philipp Reinecke for the many little shell scripts he quickly set up and for his dedicated work in the restart project. I thank Steffen Tschirpke for technical support and Christine Henze for help in all administrative matters. I am grateful to my former colleagues at Humboldt-University for many fervid discussions that broadened my view.

Aad van Moorsel has been a colleague on the restart project for many years. He is the perfect person to write a foreword and I am happy he agreed to do so.

I want to take the opportunity to thank Jochen Schiller and his group for giving me such a warm welcome and making me feel at home immediately at Freie Universität. This made me anticipate future work related to restart and also broadened my horizon.

I would like to thank Springer Verlag and Ralf Gerstner in particular for agreeing to publish this book.

Finally, the most crucial support throughout the past years came from my family. I thank Tobias Zepter and our children Louise, Margarete, Elisabeth, Samuel and Jakob just for being there and being my delight every day.

Berlin, Germany, Katinka Wolter
October 2009

Contents

Chapter 1
Basic Concepts and Problems

This book adresses problems and questions in computer fault-tolerance that can be tackled using stochastic models. Computer fault-tolerance is an important feature in mission-critical or highly available systems. It is implemented through a system design that includes replication and redundancy [127]. We are interested in problems related to redundancy rather than replication, namely the question when to issue a certain methods. Redundancy mechanisms can be divided into redundancy in space, where a stand-by component may take over service, and redundancy in time, where the service is repeated. We will more closely consider the latter and study different methods that have in common that they all implement some mechanism of repeating a task. Recent studies [68, 117] show that modern software systems are far from being fault free and the faults causing a system failure often can be localised. Even more, the Pareto distribution of faults [58] holds, i.e. 80% of the faults are located in 20% of the files. Frequently, failure of the system is caused by a number of small faults that cannot be removed. In this situation restarting modules may be the method of choice.

Even though there are many interesting open issues in software fault-tolerance the focus of this book lies on the timeout selection problem rather than on the many other aspects of fault-tolerance mechanisms using redundancy in time. Before diving into a treatment of stochastic models adressing the timeout selection problem for the three selected methods, i.e. restart, rejuvenation and checkpointing, this chapter discusses preliminary aspects of the general timeout problem common to all further algorithms.

1.1 The Timeout Problem

Timeout problems arise virtually in all walks of life, including human-created systems such as computing systems. Every day often several times we find ourselves waiting for some service, or waiting for a delivery, wondering *how long should I wait?* In computing systems one has to wait for the result of a computation, the response to a request, or a reply in a communication. Set aside the situations where a system deliberately waits for customers. Behind the question, how long one should

K. Wolter, *Stochastic Models for Fault Tolerance*,
DOI 10.1007/978-3-642-11257-7_1, © Springer-Verlag Berlin Heidelberg 2010

wait, lies the suspicion the system may have failed, the length of a queue might have become too long such that the request may have got lost, the connection may have been disrupted or a contributing process may have erroneously stopped and will never reply. In real-life situations the length of the line in front as well as the performance of the person behind the counter can be observed. Based on both impressions the decision whether to stay in line or to drop out is taken. The human user thus applies an educated guess in order to decide whether the waiting time is still within reasonable limits of the system response time in healthy state. This guess can often be more precise in a human environment than when facing a computing system. The less is known about the internal configuration of the system, the more a user has to rely upon his feelings. Typically first the user will wait optimistically for some time followed by a period of doubt, until finally one gives up hope and cancels the session, aborts the computation, or interrupts the communication.

The question how long to wait touches on different fields of computer science. It implies consideration of the reasons for long waiting time and investigation of possible solutions of the problems leading to long waiting times. Waiting time can be long because of a failure somewhere in the system where no appropriate fault-tolerance mechanisms are deployed. One might have to wait because a computation takes much longer than anticipated, or some part of the system is just very slow, or overloaded. Analysis of delays in computing systems requires detailed consideration of system performance and reliability parameters and possible performance and reliability enhancements.

We study different performance and reliability mechanisms commonly used in computing systems. One can do this at different levels of abstraction. Implementation of the mechanisms at system level raises many questions, such as when to use a method, how to implement it, in which layer of the software stack to use what mechanism, how to evaluate each method, how to improve them and how to parameterise them. The last two of those exceed pure systems questions and are commonly answered using formal methods, or mathematical models. This work is concerned with stochastic models and stochastic modelling solutions to some particular timeout problems as they arise in performance and reliability mechanisms.

In the second part, the first technical part of this work, a black box approach is used to find ways to speed up processes. The restart method does not explicitly consider the shortcomings in a system that may lead to long waiting times. Instead, it uses an engineering perspective that does not require an understanding of the system dynamics leading to exceptionally long delays. It uses the system's external timing behaviour to, first, identify situations in which matters can be sped up by aborting a process and starting it anew, and, second, develop algorithms that determine the optimal timeout efficiently. This part consists almost exclusively in recent research contributions of the author and her colleagues.

The method studied in the third part focusses on the impact of the environment on process behaviour. It analyses not the performance of the process itself, but its response to the degradation of the environment. The process environment is analysed, modelled and tuned to improve the process performance and reliability.

Rejuvenation denotes a periodic restart of the process environment in order to increase system performance and reduce the failure probability.

The fourth part is concerned with checkpointing, which is the most fine-grained performance and fault-tolerance mechanism out of the considered approaches. Checkpointed systems periodically save the system state such that in the case of system failure no complete restart of the running processes is necessary, but all processes can be reset to the most recent checkpoint and continue processing from there. The checkpoint intervals can be placed in different ways depending on the considered system and the metrics of interest.

At first sight the mechanisms discussed in this work, restart, rejuvenation and checkpointing, seem to be very similar. In all approaches processing is interrupted and started again. All three methods use a time interval, either as a timeout for process or system restart or for the placement of checkpoints. But even though all methods relate to the general timeout problem each one of them does so in a different way. When using the restart method completion of a process is anticipated as soon as possible and action is taken if the result is not delivered within a given time, whereas rejuvenation and checkpointing are applied to extend the time until system failure. Rejuvenation and checkpointing also speed up task completion but they do so indirectly by avoiding system outage. When applying the restart method not the environment is observed but the considered task. Rejuvenation, in contrast, requires monitoring the system, not the processes it executes and is applied to the system not to the processes. Checkpointing, finally, combines properties of both, restart and rejuvenation. To apply checkpointing the system failure behaviour and the task processing must be observed. The rollback to the most recent checkpoint can be applied to either the task, the system or both. As with rejuvenation the purpose is to circumvent system failures and achieve completion of processes in as short a time as possible. However, the most important difference between restart and both rejuvenation and checkpointing is that the former relates to a minimisation problem with known limit of the optimum, while both latter methods relate to unbounded maximisation problems.

Just as timeout problems in systems at first glance seem to be all the same, the stochastic models appear to be very similar. Only a deeper analysis of the stochastic models formulated for restart, rejuvenation and checkpointing reveals the great impact of the small differences, as will become clearer throughout this work. Analytical models for retrial queues [42, 43, 6] can be seen as more general predecessors the models considered in this text.

A profound understanding of timeout problems in general may provide us with a tool box of appropriate solutions which can be applied after checking some characteristics of the given system. This book takes one first step towards the necessary appraisal of timeout problems by summarising and comparing different timeout mechanisms.

Many threads of the work in this book may be further expanded as they require a separate tractive. Especially development of systems that implement restart, rejuvenation and checkpointing, but also evaluation of such system would be of utmost