A Short Investigation into an Underexplored Model for Retries

Aad P. A. van Moorsel
School of Computing Science
University of Newcastle upon Tyne
Newcastle upon Tyne, NE1 7RU
United Kingdom
Aad.vanMoorsel@newcastle.ac.uk

and

Katinka Wolter
Humboldt-Universität Berlin
Institut für Informatik
Unter den Linden 6, 10099 Berlin,
Germany
wolter@informatik.hu-berlin.de

Abstract

We provide a short overview of models being used for the performance analysis of retry mechanisms in software systems, including checkpointing, preventive maintenance, rejuvenation, restarts and time-outs. We relate this to a simple model we recently analysed, and hypothesise why this particular model has not been analysed in detail before.

1 Introduction

Several methods for software performance and reliability enhancements have been investigated over the last decades, including checkpointing, software rejuvenation, preventive maintenance and restart. For those mechanisms, many different stochastic models have been proposed and analysed. These models have a lot in common, since the mentioned mechanisms have many similarities. All deal with systems that operate in an environment that is subject to maintenance and failures (and associated repairs), so that processing is interrupted and restarted, the latter possibly from an intermediate checkpoint.

Recently, we provided a detailed analysis of 'yet another' model [6], namely one for restart, suitable for scenarios with highly variable response times. The model has a striking simplicity, and the question arises why the model had not been analysed before. In this paper, we try to answer that question by discussing the modelling literature for systems with retries in light of our restart model.

2 Basic Restart Model

Our model is very simple [6]. A task is started, and when it has not completed at a threshold time, it is retried. The task is assumed to complete according to some probability distribution, and it is assumed that each retry terminates the previous attempt. The question is: in order to minimise the completion time, what is the best time to restart?

We found a number of elegant and interesting results. A restart should not necessarily take place at 'fixed' times between successive tries. Only when one wants to minimise the first moment of completion time, the best strategy is to retry at fixed intervals. In particular, the retry should take place at the time point where the hazard rate is reciprocal to the inverse of the resulting completion time. For higher moments of completion time, it is better to initiate restarts at a fast pace at the beginning, and then slow down. For the distribution of the completion time, also interesting results hold. For instance, to maximise the probability of making a deadline, one should do restarts at time points at which the hazard rates are equal. A special case that obeys this criterion is to restart at equi-distant time points, but this is not always the global optimum (it could in fact correspond to a local or global sub-optimum).

3 Time-Out Models

A closely related problem is that of setting time-outs, such as time-outs in the TCP transport protocol [3], or in failure detectors (see [6]). Extensive research has gone into practical solutions for determining a reasonable TCP time-out as a function of the expected round trip time (i.e., completion time). Exponential back-off is then employed to make sure that one does not endlessly retry too quick.

In a similar vain, models for restart that do not make any assumption about the distribution of completion time have been developed (see [6]. It has been shown that a back-off mechanism that is not dissimilar to exponential back-off is of optimal order, no matter what the actual distribution of response time is. A core assumption in this work is that consecutive tries are statistically independent and retries abort earlier attempts. This work does not consider the 'network effect', which occurs when many tasks follow the same strategy. Instead, all analysis is done for one isolated end-point, assuming all other systems to behave as before. Only Huberman conducts a discrete-event simulation to study the network effect (see [6].

For the quality of service of failure detectors, a secondary issue comes into play: when retrying, the previous attempt is not cancelled. If scarceness of

resources is not modelled, the optimal strategy is to continuously retry. In a real system, this obviously leads to bandwidth and CPU overload. The models considered by Chen *et. al.* therefore include bandwidth concerns (see [6].

4 Checkpointing and Rejuvenation

Checkpointing models [1, 5] typically assume that a fixed amount of work needs to be performed in an environment that is subject to failures. Many of the models include state information that describes whether the system is operational or in repair. The models vary in conditions upon the state model: some limit the number of visits to the failed state, others limit the total time spent in the failed state to represent the processing of critical real-time tasks. Upon failure of the system either no work is saved, all work is saved, or some intermediate amount is saved, representing the placement of checkpoints. The models furthermore differ in whether they allow for failures during checkpointing or not. A special case is the one where checkpointing at zero cost is performed. The optimisation problem then is to find the best checkpoints such that the probability of completing a given amount of work in the given environment is maximised.

A checkpointing model with immediate repair and repeat strategy (all work is lost) is very similar to the restart model. The only difference is that in the restart model the amount of work is a random variable whereas in the checkpointing literature it typically is a constant.

Software rejuvenation models [1] typically assume, similar to the checkpointing models, a fixed amount of work to be completed. However, the metric of interest usually is not completion time, but system availability. Software rejuvenation models have a rather sophisticated state description. The system may be up or down, and it can be down after a failure or out of order due to rejuvenation tasks. It is assumed that rejuvenation takes less time than repair and the problem is to determine how often the system should be rejuvenated such that the overall system availability is maximised and system down time is minimised.

The main difference between models for rejuvenation and restart is that the rejuvenation models include system detail to describe the aging or degradation aspect of the system, while the restart model includes no system information. In both type of models job processing is interrupted, but in the rejuvenation model this is a side effect of rejuvenation, while for the restart model it is the essential model behaviour.

Checkpointing and rejuvenation can be combined, the former operating on the software and task operation and the latter improving system reliability and performance. In this case the considered metric is the moments of task completion time, as typical in checkpointing models.

5 Preventive Maintenance and Survival Analysis

Preventive maintenance models [2] aim at maximising system operation time, which leads to an optimisation problem that is almost the dual to that considered in restart, time-outs and checkpointing (and to some extend rejuvenation). Hence, a strategy that made sense for restart would have the opposite effect in preventive maintenance. Nevertheless, the mathematics is essentially identical (always related to setting derivatives to zero) and the results should therefore translate between the two opposed objectives.

A main difference, however, is the following. When minimising the completion time, it is natural to add a time penalty in the model every time a restart takes place (every restart kills the previous attempt and triggers the next attempt, and this takes time). However, when maximising system life time, such a time penalty does not make sense, since it only enlarges the life time. This would imply that very long and continuous maintenance is best for the system, something that does not correspond to real life. As a consequence, preventive maintenance models introduce costs into the model to represent a penalty associated with triggering maintenance.

Survival analysis [4] deals with estimation of the success of medicine on the life time of humans (or, in fact, other species). The focus in this work is typically on statistical estimation of hazard rates, which play a crucial role in characterising the life time of a subject. The importance of the hazard rate comes forward in restart analysis as well, but the survival analysis literature rarely concerns itself with models for optimal timing of certain actions, probably since providing medicines is not an instantaneous activity.

6 Conclusion

We have given a brief overview of various models that deal with the effect of retries on task completion time. We did this in light of a recent model we analysed, and which, much to our surprise, had not been analysed in detail before. In this write-up, it becomes clear that the model possibly never was considered useful. First of all, the analysis only leads to non-trivial results if the completion time distribution has high variance (such as in the Internet), or is a defective distributions (modelling failed attempts). Furthermore, in preventive maintenance models, the 'dual' objective makes it impossible to include penalties as easily as in the restart model. Hence, the preventive maintenance literature is concerned with

models that include maintenance costs. In checkpointing and rejuvenation, the simple model we discuss comes forward as a special case, but it never has been analysed in detail. This may be because different metrics are of interest, or because more aspects of the system are considered important, such as the system aging phenomenon in rejuvenation.

Note: much more relevant literature could be listed in this paper. For this short contribution, we do not provide all references, but instead point the reader to the references in the following articles.

References

[1] A. Bobbio, S. Garg, M. Gribaudo, A. Horvath, M. Sereno and M. Telek, "Modeling Software Systems with Rejuvenation, Restoration and Checkpointing through Fluid Stochastic Petri Nets," *IEEE Petri Nets and Performance Models,* pp. 82–91, IEEE Computer Society Press, 1999.

[2] I. Gertsbakh, *Reliability Theory, With Applications to Preventive Maintenance,* Springer Verlag, Berlin, 2000.

[3] B. Krishnamurthy and J. Rexford, *Web Protocols and Practice,* Addison Wesley, 2001.

[4] J. Lawless, *Statistical Models and Methods for Lifetime Data,* John Wiley and Sons, 1982.

[5] V. Nicola, "Checkpointing and the Modeling of Program Execution Time," in *Trends in Software 3: Software Fault Tolerance*, M. Lyu (Ed.), Wiley & Sons, Chichester, UK, Chapter 7, pp. 167–188, 1995.

[6] A. van Moorsel and K. Wolter, "Analysis and Algorithms for Restart," *IEEE Quantitative Evaluation of Systems,* IEEE Computer Society Press, pp. 195–204, 2004.