

# Self-Rejuvenation - an Effective Way to High Availability

Mirosław Malek, Felix Salfner and Günther A. Hoffmann

Humboldt-Universität zu Berlin

{malek | salfner | gunho@informatik.hu-berlin.de}

## **INTRODUCTION**

Most computer users know that one of the most effective ways to recover from faults such as system crashes and performance degradation is simply to reset or reboot a computer. Although these methods might be expensive in terms of downtime, they are frequently most effective. Since commercial users in most cases cannot afford the restart, they tend to use periodic rejuvenation, which is restarting of processes, programs or entire systems in a preventive manner at the appropriate time for a given application.

We propose self-rejuvenation, i.e., automatic rejuvenation based on failure prediction by means of modeling with Markov chains and Universal Basis Functions (UBF). Once the severity of a potential failure is determined by our models, the rejuvenation procedure may begin either by a process restart, a restart of application, reboot of the system, check-pointing, going back to recovery point or fail-over by reloading and recomputing an application on another computer. There is a strong correlation between failures and system dynamics. We show how system models can be constructed by analyzing error logs and ‘fever charts’ from system’s logs. Furthermore, we demonstrate the quality of our models based on real-system data. It will be shown that prediction-triggered self-rejuvenation is an effective way to reducing unavailability by an order of magnitude.

## **RELATED WORK**

Although rejuvenation has been informally used for years, the first formalized publication dates back to 1995 [1]. Since then its theoretical properties have been studied extensively using various methods (see [2] for an example) and were proposed in Recovery Oriented Computing [3]. Research carried out in [4] shows that prediction based software rejuvenation outperforms time estimation based methods. Our approaches are in the spirit of measurement-based models [5]. However, we do not apply trend analysis to continuous resources but use pattern recognition and function approximation techniques.

## **MODELING TECHNIQUES**

We correlate recordings of errors and additional continuous system variables such as memory usage or operating system workload with failures (e.g., process, component or system failures). The available system data is characterized by event driven log file entries and time continuous variables. Two modeling techniques were employed, each one suited for a particular data type. We use a Markov chain based approach to build models based on log file entries and Universal Basis Functions (UBF) to express continuous system variables.

### **Markov Chain Approach**

Occurrences of same failure type are aligned such that the failure event occurs at  $t = 0$ . Beginning from  $t = 0$ , patterns of events are extracted using a hierarchical clustering technique and additional algorithms to calculate the optimum

number of clusters proceeding backwards into history before the failure occurred. Each cluster forms a state in the resulting Markov chain (see Figure 1). The distinctiveness of our method is the construction of states: In contrast to [5], the ‘coding’ of the state itself comprises the short history of the system and the estimated time to failure yet preserving the Markov property which enables us to use simple Markov chain analysis techniques.

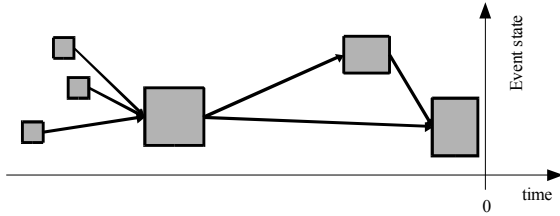


Figure 1: Model for one failure  $F$

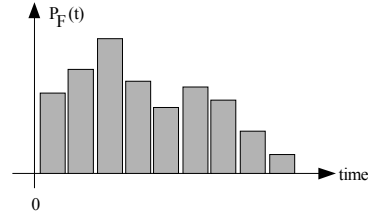


Figure 2: Probability of failure  $F$  at time  $t$  in the future

Approximation of transition probabilities by additionally analyzing log data without failures constitutes the second phase of model generation. A simple discrete time Markov chain (DTMC) is generated where end nodes indicate ‘failure’ and ‘no failure’ states. By calculating absorption probability distributions the probability of a failure at time  $t$  can be estimated (see Figure 2).

The model’s simplicity makes it attractive to a reliability engineer who is able to manually add states in order to cover failures that had not been captured by the data.

### Universal Basis Functions (UBF)

To model continuous variables we employ a novel data-based modeling approach Universal Basis Functions (UBF) which was introduced in [7]. UBF models are a member of the class of non-linear non-parametric data-based modeling techniques. UBF operate with linear mixtures of bounded and unbounded activation functions such as Gaussian, sigmoid and multi quadratics. Non-linear regression techniques strongly depend on architecture, learning algorithms, initialization heuristics and regularization techniques. UBF address many of these issues. The type of adaptive kernel function used in UBF can be adapted to evolve domain specific transfer functions. This makes them robust to noisy data and data with mixtures of bounded and unbounded decision regions. UBF produce parsimonious models which tend to generalize more efficiently than comparable approaches such as Radial Basis Functions [7]. We employ UBF as a function approximation scheme and as a classifier, modeling and predicting event classes indicating either a ‘failure’ or ‘no failure’ state.

### SELF-REJUVENATION

In the era of ‘self-\* computing as heralded by, e.g., IBM as autonomic computing [6] self-rejuvenation is a special case of self-repairing or self-healing. Combining a failure predictor with rejuvenation can also be incorporated into existing restart/rejuvenation architectures such as Recovery Oriented Computing. A failure predictor simply represents a monitoring agent with more sophisticated output than the ones described in [3] providing the recovery manager with much more detailed information about the system’s state. Existing rejuvenation architectures mainly profit from the information, *when* a crash occurs.

In addition to this straightforward implementation our models establish a basis for selecting the appropriate candidate and the appropriate method of rejuvenation. A candidate could, for example, be a node in the reboot tree proposed in [3] or some set of components in a component framework. Choosing a method may imply, e.g., going back to a recovery point or redeploying the entire component.

## MODELING RESULTS

Both modeling techniques presented in this paper have been applied to data of a commercial telecommunication platform. The primary objective was to predict failures in advance. We measured test data of a two node cluster non-intrusively using existing logfiles: Error logs served as source for event triggered data and logs of the operating system level monitoring tool SAR (*System Activity Reporter*) were used for time continuous data. We calculated precision, recall and F-measure [8] to evaluate the quality of our models. We compare our approaches to a naive model which was calculated by using mean-time-between-failures (MTBF). We use a lead time [9] of one minute for the DTMC models and five minutes for the UBF models. The DTMC model was built with 30 seconds of historic data, the UBF models where fed with a five minutes history. We call this data history “embedding dimension”.

Model	Type of Data	Lead time [min]	Embedding dimension	Precision	Recall	F-Measure
DTMC	log files	1	30 s	0.8000	0.9230	0.8571
UBF	SAR	5	5 min	0.4912	0.8295	0.6088
Naive		5	none	0.2500	0.2000	0.2222

Table 1: Precision, recall and F-Measure for discrete time Markov chain (DTMC), universal basis function approach (UBF) and naive model. In the case of UBF we report mean values.

## REDUCING UNAVAILABILITY

Forecasting failures can help to improve availability by increasing mean-time-to-failure (MTTF). Based on the results reported in Table 1 we calculated the effects of model-triggered self-rejuvenation on MTTF and unavailability respectively. We get values for precision and recall of 49% and 82% for a five minute lead-time. Assuming that a hypothetical system exhibits one tenth of an hour (six minutes) downtime per 1000 hours of operation (which corresponds to a “four nines” availability and 0.0001 unavailability respectively) self-rejuvenation would increase MTTF from 999.9 hours to 5555 hours of operation. Given the increased MTTF we get:

$$U = 1 - \frac{MTTF}{MTTF + MTTR} = 1 - \frac{5555}{5555 + 0.1} \approx 0.000018$$

where *MTTR* represents Mean-Time-To-Repair and *U* denotes unavailability. This is close to an order of magnitude improvement in comparison to a case without model-triggered self-rejuvenation. In complex real systems, failure

prevention will, of course, not be able to avoid all failures even if they had been predicted correctly. Even worse, false alarms may evoke additional failures.

Our results indicate that fault prediction-based self-rejuvenation might have a significant impact on availability improvement and should be a standard feature in the future generations of computer systems.

## **REFERENCES**

[1] Y. Huang, C. M. R. Kintala, N. Kolettis, N. D. Fulton: Software Rejuvenation: Analysis, Module and Applications. FTCS-25 1995: 381-390

[2] S. Garg, A. Puliafito, and K. S. Trivedi, "Analysis of Software Rejuvenation Using Markov Regenerative Stochastic Petri Net," Proceedings of the Sixth International Symposium on Software Reliability Engineering, Toulouse, France, October 1995, pp. 180-187.

[3] G. Candea, J. Cutler and A. Fox., Improving Availability with Recursive Microreboots: A Soft-State System Case Study, In Performance Evaluation Journal, vol. 56, March 2004

[4] K. Vaidyanathan, R. E. Harper, S. W. Hunter and K. S. Trivedi, Analysis of Software Rejuvenation in Cluster Systems. In Proc. of the Joint Intl. Conference on Measurement and Modeling of Computer Systems, ACM SIGMETRICS 2001/PERFORMANCE 2001, Cambridge, Massachusetts, June 2001.

[5] K. Vaidyanathan and K. S. Trivedi. A Measurement-Based Model for Estimation of Resource Exhaustion in Operational Software Systems. In Proc. of the Tenth IEEE Intl. Symposium on Software Reliability Engineering, pages 84-93, Boca Raton, Florida, November 1999.

[6] J. O. Kephart and D. M. Chess, The Vision of Autonomic Computing, IEEE Computer, Jan. 2003

[7] Hoffmann G. A. (2004); Adaptive Transfer Functions in Radial Basis Function Networks; in Lecture Notes in Computer Science LNCS by Springer; (ICCS2004), June 2004, Krakau, Eds.: Bubak M., Dick van Albada, Peter M.A. Sloot, Jack J. Dongarra

[8] Hoffmann G. A., Salfner F., Malek M. (2004), Advanced Failure Prediction in Complex Software Systems, SRDS 2004, 23<sup>rd</sup> Symposium on Reliable Distributed Systems, submitted for publication, October 2004