The Influence of Layered System Structure on Strategies for Software Rejuvenation

Olivia Das, C. Murray Woodside
Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada
email: odas@sce.carleton.ca, cmw@sce.carleton.ca

Abstract

Hardware components exhibit degradation faults. Similar to hardware components, several recent studies have described the phenomenon of "software aging" where the performance and reliability of the software system degrades with time. In an attempt to decrease the unplanned outages due to aging, a proactive technique called "software rejuvenation" can be employed. This technique involves stopping the running software, cleaning its internal state and restarting it. This extended abstract describes avenues for evaluating the effectiveness of software rejuvenation for multi-layered systems, where the failure of a service depends on other services in lower layers.

1. Introduction

Software aging [1] is a phenomenon where a software exhibits increasing failure rate and performance degradation over time. The primary causes for this degradation are exhaustion of system resources, data corruption and accumulation of errors. Software rejuvenation [1] is a proactive technique devised for reducing the probability of unpredicted system outages due to aging. This technique involves halting the running software, cleaning its internal state and restarting it. Examples of cleaning the internal state might be garbage collection, flushing operating system kernel tables and reinitializing internal data structures.

Distributed software systems are usually structured in layers with some kind of user-interface tasks at the topmost layer, making requests to various layers of servers. Examples include telecommunication systems and banking systems. In [2], an approach was developed to express layered service failure and repair dependencies, and [3] provided an efficient algorithm for computing performability measures.

It would be valuable to include the effects of software aging and rejuvenation on the failures of servers, which can induce failures of applications at higher layers. Servers in this context can also include processors. Thus, this abstract extends the model of layered systems [3], with:

- aging for hardware as well as software components. We assume that aging affects both performance and failure rate of components.
- effect of software rejuvenation.

This work is similar to the models proposed in [1, 4, 5, 6, 7] in that it uses hypoexponentially distributed (which is a increasing failure rate distribution) time to failure for modeling aging. Generally distributed time to failure and service rate being an arbitrary function of time was considered in [8] for transaction-based systems with a single queue and [6] analyzed software rejuvenation for cluster systems. Our work is different since we model aging in a layered system with multiple layers of queues.

This work assumes a prediction-based rejuvenation policy as considered in [6, 7] where the rejuvenation starts whenever a degraded state of the component is detected by means of analyzing some observable symptoms. Otherwise, the component eventually goes to an undetected failed state that usually requires higher detection and repair time than the rejuvenation time.

2. Layered Model

2.1. Model for individual components

Figure 1 shows the CTMC model for a processor. Each processor initially starts in a working state (W). As time advances, it transits to a "failure probable state" (FP) with rate λ_{1p} and eventually fails (state F) with rate λ_{2p}. There is also a transition from working state (W) to completely failed state (F) with rate λ_p to model random failures caused by random events such as sudden environmental shocks etc. A failed processor can be detected and repaired with rate μ_p. Rejuvenation is not

applied to processors.

Figure 2 shows the CTMC model for a software task, meaning an operating system process. Each task has the same model as a processor except that its degraded state (FP) can be detected with mean detection time $1/\delta_t$ by analyzing certain symptoms. After detection, it transits to state "being rejuvenated" (BR) where the rejuvenation starts with rate μ_{2t}. Otherwise, it transits from state FP to the completely failed state (state F) with rate λ_{2t}. In case of a complete failure, the task can be repaired with rate μ_{1t}. Since rejuvenation time is usually much smaller than the repair time, thus, $\mu_{1t} << \mu_{2t}$.

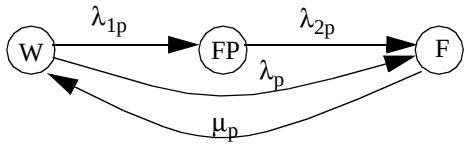

Figure 1. CTMC model for each processor.

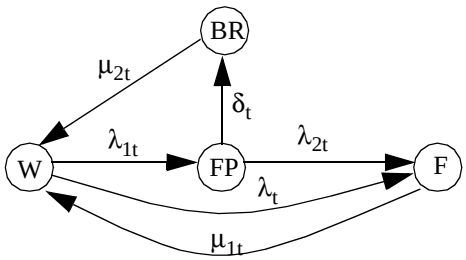

Figure 2. CTMC model for each software task.

The above CTMC's can be solved to obtain the probabilities of the component being in each of its states.

When a component is in state FP, its performance parameters are degraded but it is still operating. In state F (and BR in case of software task), the component is not operational. While a component is in one of these states, it is possible for the system to maintain operation by taking advantage of the standbys or active replicas, if any available.

2.2. System-level model

Figure 3 shows a typical example of a layered model proposed in [2, 3]. This model is an extension to a pure performance model, called Layered Queueing Network (LQN) [9], by adding dependability related parameters, i.e. the redundancies and the constant failure and repair rates for components, to it. The main difference between a layered queueing model and a queueing network model is that a server to which service requests are arriving and queueing for service might make calls to other servers, thus giving rise to nested services. This figure has one client *task*, Task 1, at the top and three layers of servers, Task 2-5, the intermediate layers and Task 6, the bottom layer. A task offers one or more services through methods called *entries* (e.g. Task 1 has entry Entry 1), and these entries execute on its processor and make calls to other entries in lower layers. An oval represents a processor. Each processor and a task has its own queue for arriving service requests. The arrows designate service requests from one entry to another, with an implied reply. Tasks block to receive the reply, as in standard remote procedure calls. The alternative targeting of requests for Service 1, needed by Entry 2, to Entry 3 or 4 or 5 is indicated by labels "#n" on arcs showing the priority of the targets. A service request always goes to the highest-priority available server.

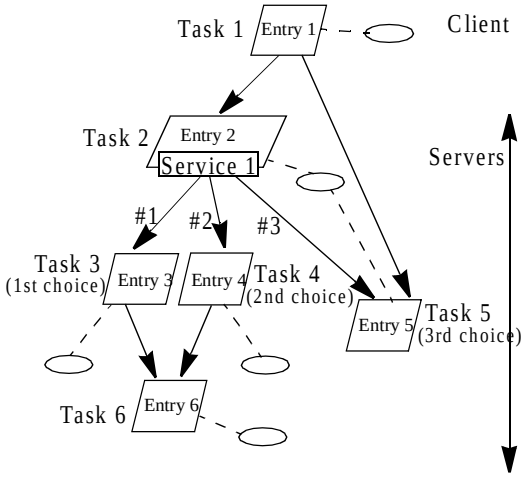

Figure 3. A Layered Model with three layers. Alternative targets for Service 1 are Tasks 3-5 in increasing order of preference.

The performance parameters provided to this model are:

- the mean CPU demand per invocation for each entry,
- the mean number of calls from an entry to other entries.
- service rate for each processor.

Aging of a processor is modeled by making its service rate a decreasing function of its state. Aging of a task is modeled by making the mean CPU demands of its entries a decreasing function of its state.

In this layered model, each combination of states of the components yields a different performance model,

also called an *operational configuration*. Each operational configuration corresponds to an LQN model and can be solved for various performance measures, for example, mean throughput, mean response time etc.

3. Model Solution

The general strategy of solving the layered model is to:

- determine all the operational configurations,
- compute the performance for each configuration by applying Layered Queueing Analysis [10],
- compute the probability of occurrence of each configuration from its component state probabilities and then
- combine the probabilities and the performance to find average performance measure for the system.

This strategy is similar to the Dynamic Queueing Network approach given in [11] for queueing network models.

3.1. Explosion of Operational Configurations

One serious drawback of the model solution is the enormous number of operational configurations due to two working states of each component. This drawback could be eased by approaching the problem from different perspectives. Some of them are as follows:

- If we assume that aging only affects failure rate and degradation in performance is small enough to ignore, then both the states "W" and "FP" would have same performance parameters. This assumption reduces the number of operational configurations to a great amount since many combinations of components states can be aggregated to one operational configuration. For this case, we can apply the methods described in [3] to find the probabilities of the configurations.
- Another possibility might be to explore symmetry in redundancy. In that case, using the primary or the backup gives rise to the same performance model and thus the number of operational configurations can be reduced.
- Another way is to approximate the solution by considering only those components that affects the system in a greater amount and ignoring others. For example the components which provide shared services are usually more sensitive than others since their failure triggers multiple failures. Identification of these components might be done by performing sensitivity analysis.

4. Example

In order to demonstrate our analysis strategy, let us consider a simple client-server system (as shown in Figure 4) with one client task, Client, with entry c1. c1 makes in average, 2 requests to entry s1. The CPU demand for entries s1 and s2 are 1 and 1.5 secs respectively. The performance is degraded by 50% for both tasks Server1 and 2 in the "failure probable" state. If Server1 fails or is rejuvenating, the client uses Server2. Let us assume $1/\lambda_{1t}$, $1/\lambda_{2t}$, $1/\lambda_t$, $1/\mu_{1t}$, $1/\delta_t$ and $1/\mu_{2t}$ for both the tasks Server1 and 2 to be 10 days, 10 days, 40 days, 1 day, 10 minutes and 1 hour respectively. Also assume that all the processors (not shown here) and the client task do not age and are perfectly reliable.

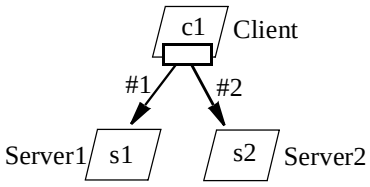

Figure 4. A simple client-server system

There are four operational configurations for this system:

- C_1: Client using Server1. Server1 is in state W.
- C_2: Client using Server1. Server1 is in state FP.
- C_3: Client using Server2. Server2 is in state W.
- C_4: Client using Server2. Server2 is in state FP.

The probabilities of the configurations and their rewards is shown in Table 1 for two cases, one involving rejuvenation of both Server1 and 2 and the other without rejuvenation.

Table 1: Probabilities and Rewards for Operational Configuration

Configuration C_i	Probability of Configuration		Reward = Throughput of Client task
	(with rejuvenation of Server1 and 2)	(without rejuvenation)	
C_1	0.9709	0.47	0.5
C_2	0.0006	0.47	0.25
C_3	0.0275	0.0282	0.333

Table 1: Probabilities and Rewards for Operational Configuration

Configuration C_i	Probability of Configuration (with rejuvenation of Server1 and 2)	Probability of Configuration (without rejuvenation)	Reward = Throughput of Client task
C_4	0.00001	0.0282	0.166
System failed	0.00099	0.0036	0

The average throughput for Client task is 0.49 requests/sec and 0.36 requests/sec for with and without rejuvenation cases. We see that rejuvenation did improve the throughput of the system significantly.

5. Conclusion

This extended abstract has described an approach for including the effect of aging and software rejuvenation in a layered performability model. The main strength of our model, is its capability of capturing the aging in a system with layers of queues. One limitation of our model might be the explosion in the number of operational configurations. This limitation can be alleviated in various ways that we have described in our third section.

References

[1] Y. Huang, C. Kintala, N. Kolettis and N. D. Fulton, "Software Rejuvenation: Analysis, Module and Applications", in *Proc. of 25th Symposium on Fault Tolerant Computer Systems*, California, June 1995, pp. 381-390.

[2] C. M. Woodside, "Performability modelling for multi-layered service systems", *Third International Workshop on Performability Modeling of Computer and Communication Systems* (PMCCS-3), Illinois, Sept. 1996.

[3] O. Das and C. M. Woodside, "Evaluating layered distributed software systems with fault-tolerant features", *Performance Evaluation*, 45 (1), 2001, pp. 57-76.

[4] S. Garg, A. Puliafito and K. S. Trivedi, "Analysis of Software Rejuvenation Using Markov Regenerative Stochastic Petri Net", in *Proc. of the Sixth Intl. Symposium on Software Reliability Engineering*, France, October 1995, pp. 180-187.

[5] S. Garg, Y. Huang, C. Kintala and K. S. Trivedi, "Time and Load Based Software Rejuvenation: Policy, Evaluation and Optimality", *Proc. of First Fault Tolerant Symposium*, India, December 1995.

[6] K. Vaidyanathan, R. E. Harper, S. W. Hunter and K. S. Trivedi, "Analysis and implementation of software rejuvenation in cluster systems", *SIGMETRICS/Performance 2001*, pp. 62-71.

[7] V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter K. S. Trivedi, K. Vaidyanathan and W. P. Zeggert, "Proactive Management of Software Aging", *IBM Journal of Research and Development*, 45 (2), 2001, pp. 311-332.

[8] S. Garg, A. Puliafito, M. Telek and K. S. Trivedi, "Analysis of Preventive Maintenance in Transactions Based Software Systems", *IEEE Trans. on Computers*, 47 (1), January 1998, pp. 96-107.

[9] C. M. Woodside, J. E. Neilson, D. C. Petriu and S. Majumdar, "The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software", *IEEE Trans. on Computers*, vol. 44, no. 1, January 1995, pp. 20-34.

[10] G. Franks, S. Majumdar, J. Neilson, D. Petriu, J. Rolia, and M. Woodside, "Performance Analysis of Distributed Server Systems", in the *Sixth International Conference on Software Quality* (6ICSQ), Ottawa, Ontario, 1996, pp. 15-26.

[11] B. R. Haverkort, I. G. Niemegeers and P. Veldhuyzen van Zanten, "DYQNTOOL: A performability modelling tool based on the Dynamic Queueing Network concept", in *Proc. of the 5th Int. Conf. on Computer Perf. Eval.: Modelling Techniques and Tools*, G. Balbo, G. Serazzi, editors, North-Holland, 1992, pp. 181-195.