ABSTRACT

In this paper we construct an evaluation framework for a self-healing system, VM-Rejuv – a virtual machine based rejuvenation scheme for web-application servers – using simple, yet powerful, probabilistic models that capture the behavior of its self-healing mechanisms from multiple perspectives (designer, operator, and end-user). We combine these analytical models with runtime fault-injection to study the operation of VM-Rejuv, and use the results from the fault-injection experiments and model-analysis to reason about the efficacy of VM-Rejuv, its limitations and strategies for mitigating these limitations in system-deployments. Whereas we use VM-Rejuv as the subject of our evaluation in this paper, our main contribution is the demonstration of a practical evaluation approach that can be generalized to other self-healing systems.

Categories and Subject Descriptors

C.4 [Performance of Systems]: Reliability, availability and serviceability

General Terms

Measurement, Performance, Reliability

Keywords

Markov chain, CTMC, VM-Rejuv, Rejuvenation

1. INTRODUCTION

Self-healing mechanisms are intended to improve the reliability, availability and serviceability (RAS) of a system by enabling it to automatically detect, diagnose and repair localized hardware and software problems [8]. However, the inclusion of recovery or repair mechanisms (self-healing mechanisms) is no guarantee that these mechanisms work well, are bug free, or that the failure modes and limitations of these mechanisms are well understood. The inadequate testing of recovery mechanisms and the unexpected/unintended negative side effects of recovery have resulted in a number of (in)famous failures, which have been discussed in previous work [2], [11], [7]. The rigorous testing, analysis and validation of these mechanisms are important but sometimes overlooked steps in system-construction that would otherwise allow designers to better understand how these mechanisms work and identify their limitations.

To assist designers and operators in system evaluations there are a number of well-studied modeling formalisms and associated analytical techniques that can be used to describe and reason about both system structure and behavior. Examples include, Markov Chains, Petri Nets, Stochastic Activity Networks (SANs), and Queuing Models ([9, 5, 12]). In terms of practical tools, fault-injection has been accepted as a powerful tool for validating and evaluating recovery mechanisms in systems [14, 1] and a number of fault-injection strategies (and tools that use them) are available [6]. Note that while fault-injection is accepted as a powerful system-validation tool it is also accepted that fault-injection cannot predict actual availability or mean time between failures (MTBF) [6]. However our goal in this paper is not to make absolute predictions about these measures, but rather to present a consistent framework for reasoning quantitatively about the limitations of recovery mechanisms and developing contingency plans that can address them.

The main contribution of our work is to demonstrate how an evaluation framework for self-healing systems can be constructed around simple probabilistic models that capture different evaluator perspectives.

2. CASE STUDY: VM-REJUV

Overview. In our case-study we model and experimentally evaluate the efficacy of VM-Rejuv – a prototype implementation of a virtual machine (VM) based software rejuvenation scheme for application servers and internet sites [13] developed at the Universitat Politècnica de Catalunya (UPC) in Barcelona. VM-Rejuv employs a prediction-based rejuvenation strategy for mitigating the effects of software aging and transient failures on web/application-servers. Software aging and transient failures are detected through continuous monitoring of system data and performance metrics of the application-server by a collection of probes; if some anomalous behavior is identified the system triggers an automatic rejuvenation action [13]. Rejuvenation actions in VM-Rejuv take the form of preventative application-server restarts.

To minimize the disruption to clients due to an application-server restart, VM-Rejuv employs redundancy and load-balancing. Web-application servers are deployed under VM-Rejuv in multiple virtual machines logically organized in a cluster consisting of a load-balancer, (which also serves as the rejuvenation coordinator), an active VM, which handles all client requests and a standby VM. When a rejuvenation action is signaled, the active VM and standby VM switch roles. New client requests are routed to the application server on the standby VM (old standby VM marked as the “new” active VM); the application-server on the old active VM finishes processing any outstanding requests before the local software rejuvenation agent (SRA) restarts the application server.

Figure 1 shows the seven node, six parameter Continuous-time Markov chain (CTMC) (from [4]) used to quantify facets of reliability, availability and serviceability for VM-Rejuv deployments.
VM-Rejuv Evaluation. We create a test deployment of VM-Rejuv running the TPC-W [10] web-application benchmark. Our deployment consists of three virtual machines: VM1 – the load balancer, rejuvenation coordinator and database server, and VM’s 2 and 3, the Tomcat application servers hosting the TPC-W web-application. VM2 is initially designated as the active VM (it handles all the incoming client requests), while VM3 is the hot standby, waiting to take over when VM2 is required to rejuvenate. See [4] for further configuration details.

In our fault-injection experiments we subject both Tomcat application servers deployed under VM-Rejuv to memory leaks that result in resource exhaustion within 5.53 minutes\(^1\) (332.017 seconds) of running a 50 client TPC-W workload. We use Kheiron/JVM [3]\(^2\) to inject memory leaks into the web-application servers running on VM’s 2 and 3. Our injection of memory leaks results in a mean rejuvenation interval of 154.06 seconds, mean rejuvenation window size of 27,401.52 msecs and mean node-failover time of 28.94 msecs. The mean time to restart Tomcat during the memory leak experiments is 3 seconds and the mean time to detect a server outage (via a heartbeat monitor) is 5 seconds.

Using Table 1 we can estimate the number of active VM failures (\(F_{act}\)) expected during rejuvenation actions per day, i.e., the frequency of transitions from \(S_1\) to \(S_5\) (\(F_{S_1 \rightarrow S_5}\)) plus the frequency of transitions from \(S_2\) to \(S_5\) (\(F_{S_2 \rightarrow S_5}\)). This we estimate at 41 per day under the failure conditions used in our experiments.

From the steady-state probabilities of the model we estimate that the deployment spends ~82% of the time in its normal operating mode/configuration, \(\pi_0\), and ~16% of its time rejuvenating (\(\pi_1 + \pi_2\)). While rejuvenations are taking place client-requests are serviced by the standby VM; as a result the system would be considered UP from the client’s perspective in states \(\{S_0, S_1, S_2\}\) – \(UP_{client} = 1416.5\) minutes per day (98.37%) and DOWN 252.5 minutes per day (17.53%), of which 229 minutes are spent performing rejuvenation actions.

Similarly, the mean time to system restoration can be quantified from the perspective of the client and the administrator. For the client, this is the mean time to restore the system to a state in \(\{S_0, S_1, S_2\}\), \(MTTSSR_{client} = 5,509\) msecs, whereas for the administrator the mean time to restore the system to \(S_0\), \(MTTSSR_{admin} = 22,373\) msecs.

3. CONCLUSIONS
In this paper we construct an evaluation framework for VM-Rejuv using simple probabilistic models (CTMCs) and runtime fault-injection tools. We use our model and experimental results to quantify metrics that can be used to reason about the efficacy of VM-Rejuv from the perspective of the designer, operator and end-user.

4. ACKNOWLEDGMENTS
The Programming Systems Laboratory is funded in part by NSF CNS-0717544, CNS-0627473 and CNS-0426623, and NIH I U54 CA121852-01A1. This work has been supported by the Spanish Ministry of Education and Science (projects TIN2007-60625).

5. REFERENCES

\(^1\)Whereas we acknowledge that a system that runs out of memory every 5.53 minutes would be quickly redesigned or abandoned by its user base, our goal is to evaluate VM-Rejuv under an aggressive memory-peak scenario.

\(^2\)Kheiron/JVM uses bytecode rewriting and the Java Virtual Machine Tool Interface to interact with running Java programs.

Table 1: Model results – VM-Rejuv steady state probabilities

\[
\begin{array}{ccc}
\pi_0 & \pi_1 & \pi_2 & \pi_3 & \pi_4 & \pi_5 & \pi_6 & \pi_7 \\
0.824573 & 0.135495 & 0.0023510 & 0.0023510 & 0.0023510 & 0.0023510 & 0.0023510 & 0.001437 \\
\end{array}
\]

\(\pi_0\) represents a window of vulnerability. From the administrator’s perspective the system is UP from the client's perspective in states \(\{S_0, S_1, S_2\}\), \(MTTSSR_{client} = 5,509\) msecs, whereas for the administrator this is the mean time to restore the system to \(S_0\), \(MTTSSR_{admin} = 22,373\) msecs.