The paper focuses on a comparative study of the main rejuvenation techniques developed so far to solve the software aging effects. These methods being studied are physical node reboot, virtual machine reboot, OS reboot, fast OS reboot, standalone application restart and application rejuvenation by a hot standby server. The paper evaluates the performance by recording and comparing the throughput loss, failed requests, slow request, and memory fragmentation overhead. The paper conducted a set of experiments injecting memory fragmentation overhead as well as analyzed the selected rejuvenation techniques' efficiency in mitigating the aging effects and the overhead of rejuvenation techniques in virtualized environment s due to the growing usage of virtualization technology. The researchers also found that the performance overheads introduced by the rejuvenation techniques are related to the granularity level. They also capture different levels of .memory fragmentation overhead induced by the virtualization demonstrating some drawbacks of using virtualization in comparison with non-virtualized rejuvenation approaches.The paper shows us that all evaluated rejuvenation strategies are similarly efficient in terms of removing the accumulated(injected) memory leak effects. We would want to use a hot-standby application server using physically different host to realize application level restart. We should avoid using virtualization methods because it would introduce a non-trivial memory fragmentation, which could cause additional aging effects since for some specific memory allocation requests, physically contiguous memory pages are required. If we have to use virtualization, we must find a way to mitigate the memory fragmentation first. For rejuvenation in OS level, if virtualization is required, we also should choose the OS fast reboot method, which has better efficiency. Besides, executing a physical machine cold reboot is better than a virtual machine. The results show that the overhead impact of the rejuvenation techniques is related to their granularity. Fine-grain techniques such as application-level rejuvenation strategies are better as a first tentative approach to mitigate the aging effects. If this first approach fails, we can use a rejuvenation technique from the next higher level of granularity. As a part of the evidence of the overhead introduced by every strategy under analysis, we observe a very important result related to the virtualization technology. Virtualization was the main cause of the drastic increase in memory fragmentation, which can cause Aging-related failures in long running systems. This means that in the long term, virtualization technology could degrade the performance of the host operating system where the hypervisor is running. This finding is especially important nowadays, since virtualization is a core technology of the new cloud-computing paradigm. Hence, selecting the right rejuvenation approach to be used in cloud-based systems in order to reduce the aging-related effects in the virtualization layer is a major requirement. Finally, we present guidelines for the use of the appropriate scenario for each rejuvenation technique under consideration.

Software aging is usually a consequence of software faults. According to system theory, a system is a collection of inter-operating elements (or components); the system boundary separates the system from its environment. For example, a single software system includes the hardware, the operating system (OS) and the applications as its elements; however, the users and other software systems are part of its environment. Each system can itself be an element of another (higher-level) system. Aging effects can also be classified into volatile and non-volatile effects. They are considered volatile if they are removed by re-initialization of the system or process affected, for example via a system reboot. In contrast, nonvolatile aging effects still exist after reinitializing of the system/process. Physical memory fragmentation and OS resource leakage are examples for volatile aging effects. File system and database metadata fragmentation are examples for non-volatile aging effects. Another example of a non-volatile aging effect is numerical error accrual preserved between system reboots via checkpoint mechanism. Note that hibernation and similar mechanisms (e.g., standby), which preserve the system memory (and thus the aging effects present in it) between system reinitialization, allow the majority of intrinsically volatile aging effects to persist even after system/process reinitialization.

Other than memory leaks, there are several other situations that prompts us to apply software rejuvenation techniques: unreleased file locks, file descriptor leaking, data corruption in the operating environment of system resources. Generally, software rejuvenation is periodic preemptive rollback of continuously running applications to prevent failures in the future. And they can be implemented in a simple application using UNIX cron command after the rejuvenation interval is determined using the analysis techniques. But we need to checkpoint the internal data structures before conducting the rejuvenation and restore after the rejuvenation. We also need a graceful shutdown and restart might be needed for rejuvenation. For those applications, we should make use of the software fault tolerance component, which is a reusable watchdog daemon process to facilitate the detection of and recovery from application process failures.

The methodologies in this paper is rather religious since they control the variants strictly and use proper number of replications to reduce experimental errors. Also, the study in the paper has chosen the most indicative factors to observe, which are throughput loss, failed requests, slow request, and memory fragmentation overhead. From these we can depict a thorough picture of what happened and what performance we get in the experiments. The results presented in the paper is informative and reasonable, which justify the correctness of our experiments design. Additionally, it would be better if the paper also keeps track of the changing process of memory fragments, so we can have some more detailed data and do some deeper analysis.

In this paper, the test program used is a multi-tier e-commerce site. For further research, we can conduct experiments on different long-time running applications and maybe on more kinds of machines and cores to testify the results we obtain from the current experiments. Also, in the previous work, we just inject one kind of memory leak in the application. We should further inject different aging factors and maybe mixing injections under variable controlling principle.