Using ML in Designing Self-healing OS

Maqsood Ahmad, Muhammad Samiullah, Muhammad Jawad
The Islamia University of Bahawalpur
Pakistan
maqsood.dba@gmail.com
samiiub@gmail.com
jawadpirzada@hotmail.com

ABSTRACT : *Operating systems serve as executing platforms and resource manager and supervisors for the applications in running phase. With the development of more complex computer systems and applications, the required operating systems become complex too. But the proper management of such complex operating systems by human beings has shown to be impractical. Nowadays, self-managing concepts provide the basis for developing appropriate mechanisms to handle complex systems with minimum human interventions. Although the implications of deploying self-managing and autonomic attributes and concepts at the application levels have been studied, their deployment at system software level such as in operating systems have not been fully studied. Self-managed applications may not enjoy the whole benefit of self-management if the platform on which they run, specially its operating system, is not self-managed. Given this requirement, this paper highlights the most frequently occurred faults and anomalies of operating systems, and proposes a tiered operating system architecture and model, and a corresponding self-healing mechanism using machine learning techniques to show how self-managing can be realized at operating system level. Based on the principles of autonomic computing and self-adapting system research, we identify self-healing systems' fundamental principles. The main objective has been to design the operating system resilient to operating system faults without restarting the operating system and less human interaction.*

Keywords : Operating System, Self-healing, Self-management Component

Received : 19 January 2016, Revised 20 February 2016, Accepted 2 March 2016

1. Introduction

Self-healing systems are becoming more popular and industry software systems and hardware architecture are being developed with self-healing properties. Nowadays, the need for more powerful, complex and reliable systems and applications are rapidly increasing to perform complex and critical tasks with in time. With the introduction of such applications, keeping them flawless, maintaining them properly, and managing them well perform and secure has become more difficult and complicated. This complication is mostly attributed to the desire for maximum availability of heterogeneous and distributed computing resources, services and mechanisms, or in a nutshell, heterogeneity challenge. The full human control over such a heterogeneous and ubiquitous world is not feasible; or better say practical. So what is the solution?

Nevertheless, there is one common underlying idea: introducing an autonomous behavior to handle an otherwise complex and unmaintainable system. This autonomous behavior must independently take decisions at runtime and manage the assigned system. Management actions (e.g., configure, adapt, recover) are goal dependent, however, must result in a consistent system [1] [2].

Autonomic computing presents the idea of reducing (ideally, removing) human interventions and making the system management autonomous and more accurate [3]. IBM [4] defines an autonomic system as a system with four basic features: self-healing, self-protecting, self-optimizing, and self-configuring.

A self-healing system is featured by being aware of system state and its ability to detect and recover from known/unknown faults. (Discover, diagnose, and react to disruptions)

A self-protecting system is a system that can detect, identify and protect the system from arbitrary attacks. (Anticipate, detect, identify, and protect itself from attacks)

Self-optimization is the ability to monitor and control the resources of a system, to get the optimal performance based on performance requirements. (Maximize resource utilization to meet end-user needs)

Self-configuration is about autonomic (re)configuration of system's components. (The ability to readjust itself "on-the fly")

Autonomic computing recommends a promising outlook for management of modern complex systems by lowering (or removing) the necessity of human interventions in the management of such systems, operating systems and applications. Self-managing applications can be invulnerable from their executing platforms if the operating systems under which they are managed are self-managed too [5]. Otherwise, applications are vulnerable to all kinds of platform errors and unwanted mistakes. It is thus necessary to have self-healing support at the operating system level as well. Finally, [8] note that exceptional situations might also require human intervention to support self-healing systems.

In this paper, we presented a model how one can apply autonomic properties in general, and specifically self-healing properties to operating systems to reduce human involvement in their management and configuration. We further propose a new three layered operating system architecture in support of self-healing attributes.

The paper is organized as follows. Second section presents the common operating system faults for which the design of self-healing operating system is proposed. In the third section some prominent related work in this field is discussed. Fourth section presented our proposed design, its efficiency and limitations. The prototyping of the proposed design is discussed in fifth section, and finally, the last section conclude the paper and discuss some future aspects of future work.

2. Operating System Faults

Generally speaking, all types of software, including operating systems and user applications, may mostly suffer from the following faults [6] [7]:

- Syntactic faults: input parameter faults etc.
- Semantic faults: inconsistent behavior and incorrect results, type checking etc.
- Service faults: QoS faults like real-time violations etc.
- Communication and interaction faults: time-out and service unavailability etc.
- Exceptions: I/O related exceptions and security related to exceptions etc

Operating system must ensure that the anomalies at application level may not be propagated to other application (or operating system). This ability of fault tolerant is recommended by our designed architecture. Therefore, the fault resiliency of operating systems is major to reliability of systems.

Soft faults are generated by the applications which in result may crash the application itself. The system can be recovered using

These applications, in most cases, without restarting the system. On other hand, hard faults are most critical which are caused by some system call accessing some memory-mapped I/O or some exception caused by the failure of some device driver; require the system to restart.

Our primarily concern is to design an architecture to detect such hard faults and then avoid the frequent occurrences automatically without restarting the whole system.

3. Related Work

This section provides a detailed description of the various approaches to self-healing system that has been adopted by researchers.

A novel software architecture model and monitoring infrastructure is proposed in [9], which can be maintained at runtime and can be used as a basis for system configuration and adaptation. The model is based on layered-approach. The task layer decides which application should be executed, based on pre-defined policies, and also sets the performance objectives and resource constraints. The system-monitoring layer incorporates the lowest level of abstractions, which monitor the system and articulate measurements and observations via a "probe bus". At the second level, a set of "gauges" are used to report information via a bus called a gauge reporting bus. A third level comprises gauge consumers which consume information broadcast by the gauges. These different layers would coordinate and combine multiple self-manageable components to maintain consistent sensing and information acquisition, and to ensure coherent decisions.

In [10] an adaptive mirroring strategy for external agents is proposed. This approach is based on formal and fragile methods for utilization of probes to intercept a system workflow and revolve data and control through a different path. These probes move data to and from an agent-based service channel that emulates the target system. This target space can adaptively shape its inner workflow; reliable data links among many points within the target application can thereby take place.

A programming paradigm based on the actions of biological cells and demonstrate the ability of systems built using the model to survive massive failures is presented by [11]. They concluded that error free programming is required for designing and developing fault tolerance traditional system, which adds significant costs and complexity to the process design (i.e.; design, implementation and testing phases). They illustrated their proposed model with experiments and applied it to design a wireless file service for ad hoc distributed wireless networks. Similar work is reported in [12] which conclude that concept of making robust software can be achieved by redundancy, where the redundancy is achieved by agents that have different algorithms but similar responsibilities.

The question addressed in the work of [13] is how to modify existing systems with self-healing, self-management capabilities. They proposed an infrastructure consisting of various layers with the objectives of examining, measuring and reporting of activity within the system. Within the execution of the legacy system among its components; analysis and interpretation of the events; and possible feedback to focus the direct but automatic reconfiguration of the executing system. Same problem in internet servers is identified in [14, 15, 16, 17, 18, 19] that they suffer from extreme peak loads. Traditional systems are capable of performing well under heavy load but failed to perform as expected when resources are given provision to support peak load. To solve this problem, component based architecture (DMonA) is proposed which control the concurrency control by recognizing I/O access patterns and to adapt cache strategies according to average throughput.

4. Proposed Approach

As mentioned earlier, the hard faults are unsafe and can interrupt the system to function as required. We proposed a model to minimize the functionality of the core of the operating system's kernel to decrease the probability of occurrences of operating system's hard faults. We have also given some tips to make this minimal kernel core as stable as possible. Our proposed architecture has a user layer, a kernel layer and a micro-kernel layer as is shown in Figure 1. The kernel core is placed in this layered architecture in support of self-healing features. The proposed architecture is developed on the basis of hypothesis that any operating system compliant to this model, must be able to heal operating system faults and anomalies inevitably without any human interference or restart of the whole system. To make this hypothesis valuable, the operating system must have the ability to detect the state when it is performing healthily and when it is performing unhealthily due to the occurrences of

operating system faults (hard faults). In case it is unhealthy, the operating system must recuperate from its current unhealthy state to a healthy state, doing necessary modifications. Keeping into consideration the above features, we presented five-phase scheme to design the model of self-healing operating system: 1) Something-amiss 2) Monitoring 3) Exploration 4) Detection 5) Healing

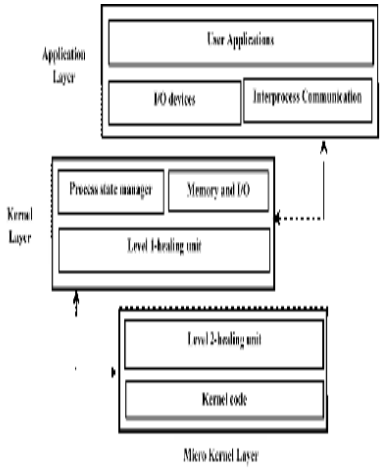

Figure 1. Proposed architecture of self-healing OS. Communication between layers is performed by system calls

In something-amiss phase the policies are discussed that how to sense something missing from the normal behavior of the system (i.e. the components are not going along the requirements, or that messages or signals are missing). Figure 2 shows the different strategies, for detecting failure by sensing something amiss from regular behavior of a system.

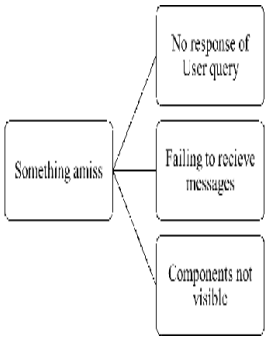

Figure 2. Something amiss phase ---- failure of receiving message, components not visible etc.

In the second phase, policies suitable for the proposed model will be discussed where the system monitors components by probing. In monitoring phase, two healing units will periodically monitor system states. The Level-1 healing unit in the kernel layer observes the application layer's activity states, and the Level-2 healing unit in the micro-kernel layer keeps an eye on the kernel activity states.

In exploration phase (the most important phase in the model) system states are analyzed in "hunting" of unhealthy and healthy states. System states are classified by some learning methods of machine learning classifiers. The Level-1 healing unit train the classifier trained with the "healthy" states of the system. The states which do not comply with the criterion of healthy states will be classified as "unhealthy" states.

In detection phase, system faults (hard faults) and anomalies are detected and appropriate strategies for their proper "treatment" are planned. Several approaches have been adopted to detect non-self or abnormality in the functioning of the system [17, 18, 19]. Various policies for detection of failure are show in Figure 3.

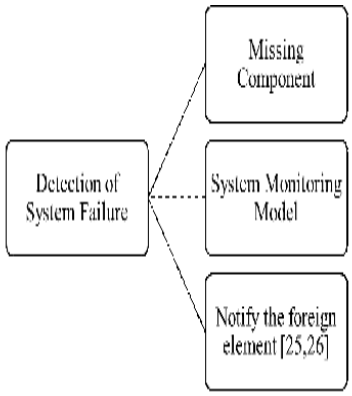

Figure 3. Detection model of hard faults during execution of the system

In last phase, healing units (Level-1 and Level-2) heal the system by transition from unhealthy states to healthy states as discussed above. Majority of the study conclude that most of the hard faults are caused by malfunctioning of the device drivers [20, 21, 22, 23], that is the reason we have chosen to move parts of kernel, such as device drivers, to user layer in our operating system architecture. Sometimes a device driver contains bugs. That bug may cause the device to malfunction (which would be the best case scenario, since usually all that happens is that the device becomes temporarily unavailable until you reboot your computer), or it can cause actual data loss. That data loss may be induced because of the bug in question corrupting data, or by a blue screen of death or panic. Yes, panic, usually an indication that the memory reserved for the operating system has been corrupted, just like when a memory module fails. With this approach we have minimized the kernel core and made the Level-1 unit responsible for healing device driver faults. Kernel faults are healed by the Level-2 healing unit in the micro kernel layer.

A number of recovery and repair mechanisms such as micro-reboot, i.e., individual rebooting of application components, without troubling the rest of the application to perform restoration in healing phase. It can achieve many of the same benefits as whole-process restarts, but for the sake of magnitude faster and with an order of magnitude less lost works. It is a fine-grain approach to overhaul the system with minimum loss. Synchronous errors such as deadlocks, and environment-dependent errors are wiped out. By deploying this low-cost recovery technique. Micro-reboot can be done at multiple levels until the faults disappear [24]. The major advantage of our proposed model persists in its nature of modularity, as a result only crashed modules (or applications) need to be boot up to recover the system. The deployment of the our proposed self-healing provides a more stable platform for running applications and can reduce stopping their execution even when they perform correctly without any faults in them.

However, it is difficult to draw a clear and vague difference between "healthy" and "unhealthy" states of a system because the transition in between the two states do not happen at discrete time. It is important that a self-healing system distinguish this progressive failure, as also identify a definite threshold to initiate rectification at, and thereby maintain its health. It is, however, a very difficult to task to detect the self and non-self both at the component level and at the system level.

5. Conclusion

In this work, we proposed a new three layered architectural model that will help in how the developers can apply dynamic properties in general, and specifically self-healing features to operating systems in order to reduce human participation in the administration and conformation of operating systems. We have also claimed that operating systems will offer a more stable execution environment for running applications after adding the self-healing property. The reason behind our claim is that the correctly behaving applications are not gratuitously and unfavorably affected by operating systems' hard faults. Although self-healing has been supported at the operating system level, a complete and comprehensive autonomic and dynamic system also

requires support for self-protection, self-configuration and self-optimization features and still many more.

References

[1] Salehie, M., Tahvildari, L. (2009). Self-adaptive software: Landscape and research challenges. ACM Transactions on Autonomous and Adaptive Systems (TAAS), 4 (2) 14.

[2] Sterritt, R (2005). Autonomic computing. Innovations in systems and software engineering, 1 (1) 79-88.

[3] Kephart, J.O., Chess, D.M.(2003). The vision of autonomic computing. Computer, 36(1) 41-50.

[4] Nami, M.R., Sharifi, M.A survey of autonomic computing systems. In Intelligent Information Processing III. Springer US, (2006). p. 101-110.

[5] David, F.M., Campbell, R.H. Building a self-healing operating system. In Dependable, Autonomic and Secure Computing, DASC (2007). Third IEEE International Symposium on, (2007) IEEE, p. 3-10.

[6] Avižienis, A., Laprie, J.C., Randell, B., Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1 (1) 11-33.

[7] Mariani, L. (2003). A fault taxonomy for component-based software. Electronic Notes in Theoretical Computer Science, 82(6) 55-65.

[8] Ghosh, D., Sharman, R.H., Rao, R., Upadhyaya, S.(2007). Self-healing systems—survey and synthesis. Decision Support Systems, 42 (4) 2164-2185.

[9] Cheng, S.W., Garlan, D., Schmerl, B., Steenkiste, P., and N. Hu. Software architecture-based adaptation for grid computing. In High Performance Distributed Computing, (2002). HPDC-11 (2002). Proceedings. 11th IEEE International Symposium on, (2002). IEEE. pp. 389-398.

[10] Combs, N., Vagle, J. Adaptive mirroring of system of systems architectures. *In:* Proceedings of the first workshop on Self-healing systems, (2002). ACM. p. 96-98.

[11] George, S., Evans, D., Marchette, S. A biological programming model for self-healing. In Proceedings of the (2003). ACM workshop on Survivable and self-regenerative systems: 10th ACM Conference on Computer and Communications Security, (2003). ACM. p. 72-81.

[12] Huhns, M. N., Holderfield, V. T., Gutierrez, R. L. Z., Robust software via agent-based redundancy. (2003) Proceedings of the second International Joint Conference on Autonomous Agents and Multiagent systems, (2003). ACM. p. 1018-1019.

[13] Kaiser, G., Gross, P., Kc, G., Parekh, J., Valetto, G. G. (2005). An approach to autonomizing legacy systems. COLUMBIA UNIV NEW YORK.

[14] Michiels, S., Desmet, L., Janssens, N., Mahieu, T., DistriNet, P.V. Self-adapting concurrency: The DMonA architecture. *In:* Proceedings of the first workshop on Self-healing systems, (2002). ACM. p. 43-48.

[15] Nagpal, R., Kondacs, A., Chang, C.(2003). Programming methodology for biologically-inspired self-assembling systems. *In:* AAAI Spring Symposium on Computational Synthesis.

[16] Valetto, G., Kaiser, G. (2002). A case study in software adaptation. *In*: Proceedings of the first workshop on Self-healing systems, ACM. p. 73-78.

[17] Aldrich, J., Sazawal, V., Chambers, C., Notkin, D. Architecture-centric programming for adaptive systems. *In:* Proceedings of the first workshop on Self-healing systems, (2002). ACM. p. 93-95.

[18] Garlan, D., Schmerl, B. Model-based adaptation for self-healing systems. *In:* Proceedings of the first workshop on Self-healing systems, (2002) ACM. pp. 27-32.

[19] Minsky, N. H. (2003). On conditions for self-healing in distributed software systems. *In:* Autonomic Computing Workshop. (2003). Proceedings of the, (2003). IEEE. p. 86-92.

[20] Herder, J.N., Bos, H., Gras, B., Homburg, P., Tanenbaum, A.S. Fault isolation for device drivers. *In*: Dependable Systems & Networks, (2009). DSN'09. IEEE/IFIP International Conference on, (2009). IEEE. p. 33-42.

[21] Chou, A., Yang, J., Chelf, B., Hallem, S., Engler, D. (2001). An empirical study of operating systems errors, ACM. 35(5) 73-88.

[22] Swift, M. M., Annamalai, M., Bershad, B. N., Levy, H. M. (2006). Recovering device drivers. *ACM Transactions on Computer Systems* (TOCS), 24 (4) 333-360.

[23] Ball, T., Bounimova, E., Cook, B., Levin, V., Lichtenberg, J., McGarvey, C., Ustuner, A. Thorough static analysis of device drivers. *In:* ACM SIGOPS Operating Systems Review, (2006). ACM. 40 (4) p. 73-85.

[24] Candea, G., Kawamoto, S., Fujiki, Y., Friedman, G., Fox, A (2004). Microreboot-A Technique for Cheap Recovery. *In*: OSDI. 4: 31.

[25] Ji, D., Dasgupta, Z., González, F. A.(2003). Artificial immune system (AIS) research in the last five years. *In*: IEEE Congress on Evolutionary Computation, (1) p. 123-130.

[26] Merideth, M. G. (2003). Enhancing survivability with proactive fault-containment. *In*: DSN Student Forum, Citeseer, 20 (3)