A Prototype Model for Self-Healing and Self-Reproduction In Swarm Robotics System

Yuan-Shun Dai[a], Michael Hinchey[b], Manish Madhusoodan[a], James L. Rash[b], Xukai Zou[a]*

[a] Department of Computer & Information Science, Purdue University School of Science, Indiana University, Purdue University, Indianapolis, IN, 46202, USA

[b] Information Systems Division, NASA Goddard Space Flight Center, Greenbelt, MD 20771, USA

Abstract

A Swarm Robotics System is a special type of wide-area and large-scale distributed system, which focuses on a group of robots cooperating to achieve the same goal using Swarm Intelligence. To treat the Swarm Robotics System with the self-healing and self-reproduction functions, this paper studied a prototype model based on the virtual neurons, autonomous self-diagnosis, consequence-oriented prescription, autonomous self-curing, and self-reproduction. This prototype system with self-healing has been implemented in the Trusted Electronics and Grid Obfuscation (TEGO) research center. Several practical cases were studied to show the effectiveness and efficiency of the model. The results demonstrate that the self-healing mechanism makes the system more reliable and the performance much improved, not only against failures, but also against failure propagations.

1. Introduction

As computer systems become increasingly large and complex, their dependability and autonomy play a critical role at supporting next-generation science, engineering, and commercial applications [1-2]. These systems consist of heterogeneous software/hardware/network components with a range of capacities and availability, and with applicability in varied contexts. They provide computing services to large pools of users and applications, and, thus, are exposed to a number of dangers such as accidental/deliberate faults, virus infections, system failures, etc., [15-16]. As a result, computer systems often fail, become compromised, or perform poorly, therefore becoming unreliable. Consequently, it remains a challenge to design, analyze, evaluate, and improve dependability.

As a special type of wide-area and large-scale distributed system, the Swarm Robotics System (SRS) represents a generic concept that focuses on a group of robots (such as aircraft) working for a mission with Swarm Intelligence [4,12]. There are many applications of such SRS, e.g., NASA's ANTS concept mission for exploring the asteroid belt using numerous pico-spacecraft [14], and other applications, such as [3, 8]. To endow such systems with the abilities of self-healing and self-reproduction is difficult but clearly beneficial. One of the solutions may reside in autonomic computing, which will be explored in this paper.

The idea of Autonomic Computing is to mimic the autonomic nervous systems found in biology, as first presented by IBM researchers [5] and in other similar research [9,10,13]. The autonomic nervous system controls the animal's body in an unconscious manner. For example, one's finger can immediately evade if it touches boiling water without conscious thinking. Some other examples of control by the autonomic nervous system include the heart beat, respiration system, blood flow, immune system, knee jerk and so on. Today's computing systems are like humans, without the autonomic nervous system, especially for large-scale systems. This is why most people feel that the Internet is unreliable and vulnerable. To solve this problem, building up a bionic autonomic nervous system for the Internet is our research focus, which is aimed at enabling self-healing against various failures or downgrade.

This paper, thus, studies a prototype system for self-healing and self-reproduction based on the virtual neurons incorporated into the SRS. The overall architecture of the virtual neuron is designed along with other autonomous diagnosis and curing modules. Based on this prototype model, the SRS could possess the self-healing and self-reproduction characteristics that were challenging topics before. The virtual neurons and self-healing function have been implemented in the TEGO research center (http://tego.iupui.edu/). Several practical cases were studied to show the effectiveness of the self-healing function, and analyzed for the influence of the novel mechanism on the malicious and benign processes. The results demonstrate that the self-healing mechanism makes the system much more reliable and the performance is much improved not only against failures but also against failure propagations.

The rest of the paper is organized as follows: Section 2 describes the overall architecture of the self-healing mechanism embedded into the SRS, and the virtual neuron is designed. Section 3 focuses on the novel functions of self-reproduction and self-healing for the SRS. Section 4 implements the self-healing mechanism into a real case, and the experimental results are analyzed to show the effectiveness of the self-healing function. Section 5 concludes this paper and indicates possible future research.

2. Overall Architecture.

The SRS is a generic concept that focuses on a group of robots (such as aircraft) working toward the same goal using Swarm Intelligence. To endow such a system with the self-healing and self-reproduction abilities is difficult but very useful. This paper finds one effective solution that is based on Autonomic Computing. The virtual neuron is presented first as the foundation, and then the system architecture is designed. Based on this overall architecture, Section 3 will depict the details on the use of these modules to realize self-healing and self-reproduction for Swarm Robotics.

2.1. System Unit: the Virtual Neuron.

The virtual neurons simulate the neurons spreading over the animal body to “perceive” the “environment” and to enable a “reflex” to “stimulations”. The virtual neuron is a tiny piece of software element that can run in the machine (such as a robot) as a background process for the purpose of perception and reflex, i.e. to “feel” context information (such as CPU speed, memory usage, processes consumption, bandwidth, communication speed, etc.). Since the size of this software element is small, it occupies minimal memory and fewer CPU cycles and thereby not affecting the overall system. Moreover, the robot’s sensors can also act for the purpose of perceiving its environment for the virtual neurons.

The virtual neurons have a similar structure even though they may play different roles. A virtual neuron has four components: Perceptron, Communicator, Recognizer and Reflexor. The perceptron gleans the information from its own monitoring and communicator (i.e., the information gleaned by other peer robots). The communicator is used for cooperation and data transmission among peers. The recognizer analyzes the information collected by the perceptron and then identifies some critical features or abnormal phenomena according to the assessment of models. The reflexor can reflex by receiving the commands from the recognizer, and execute the prescription.

Though the Perceptron can glean various information (such as CPU speed, memory usage, processes status, network traffic etc.), the context awareness is adopted here. The context information is different from the content information. For example, the data transmitted through the network; the virtual neuron does not analyze the content or meaning of the data, but is aware of the context information, such as the size of the message, the direction of the transmission, and the number of connected channels. In addition, the temperature collected by the robot’s sensor can reflect some malfunctions caused by overload.

Communicator is another component in a virtual neuron, used for cooperation among the virtual neurons in a holistic manner. Within a cell we have a head neuron which controls the other neurons and monitors that they are functioning properly. Basically the head is responsible for its cell and acts as a communicator between the cells. Each virtual neuron registers its neighbor’s virtual neurons and the head neuron within a cell. The registration is bidirectional, i.e., when a virtual neuron is turned on (usually starting together with the PC), it reports to other active neighbors in a range (such as a circle or a domain). Then, the other virtual neurons add this new one into their neighbor list, and meanwhile this virtual neuron knows all the active neighbors by their response signals. When a virtual neuron turns off, it sends a message to its neighbors for removing it from their neighbor lists. After knowing its neighbors, the virtual neurons can communicate Peer to Peer; while knowing the head, the virtual neurons can be organized in a hierarchical manner.

Recognizer operates based on models and predefined features. It analyzes the information gleaned by the perceptron and the information reported from other virtual neurons, and then identifies problems in accordance with the models and features. The recognizer may also share the complex analysis with other peer virtual neurons. Then, it can tell the Reflexor to solve the identified problems. More details on how they work for self-healing will be discussed in Section 3 and Section 4.

Reflexor can reflex by receiving the commands from the recognizer, and executing the prescription. The prescriptions can be stored locally, such as simple and often-used prescriptions, while some more complicated prescriptions can be prepared in various dedicated servers or databases, and then sent to the reflexor to execute if needed. This is because the virtual neuron should be condensed without consuming much computational or storage resources on its host robot.

The virtual neuron has been developed and implemented in our TEGO center. The Section 4 will show the implementation and case study.

2.2. Swarm Robotic System Architecture.

After developing the virtual neurons, they should be injected into the swarm of robots, and then they can form the system together with other modules for the purpose of self-healing. The architecture is briefly depicted by Fig.1.

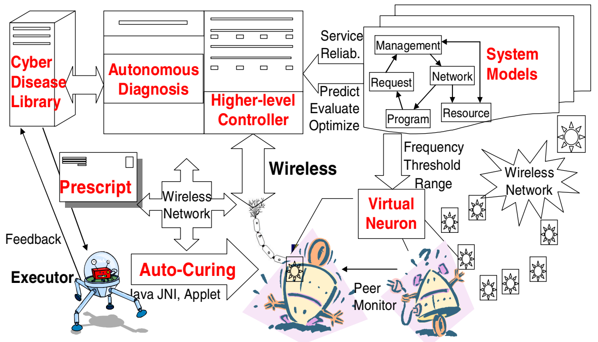

Fig. 1. Swarm Robotic System Architecture (SRSA).

There are different modules in this SRSA working together with the Virtual Neurons:
1) The system models output the normal range predicted for the corresponding subsystems, and this determines the monitoring frequency of the virtual neurons and some thresholds for reflex;
2) The virtual neurons injected into each robot glean context information. If abnormal or suspect events occur, the virtual neurons will ask the higher-level controller for more analysis and diagnosis;
3) Then, the autonomous diagnosis module works while the higher-level controller continues directing more virtual neurons to collect subsidiary information to be fed into the diagnosis module;
4) Accompanying the diagnosis, the Cyber-Disease Library is another module that has been prepared as a knowledge base which stores various symptoms and prescriptions for diagnosis and curing. The Cyber-Disease Library also supports the dynamically upgraded function for new symptoms and supports self-generated new prescriptions via machine learning technology;
5) The prescription language is systematically set up based on real-time Java to translate the curing methods into executable instructions. Based on well-defined virtual machine and class libraries, Java is inherently more portable than languages like C/C++. Java allows programs to run on a heterogeneous set of resources without any need for recompilation because it runs on its own virtual machine rather than the machine architecture itself.
6) Finally, the assigned executors will attempt to cure the patient-robot according to the received prescriptions. The executor can be the patient-robot itself if it is still operational though degraded. The executor after curing will feed back the results to the Cyber-disease library and to the system models for archiving, machine learning and model adjustment.

Under this overall architecture, the self-healing and self-reproduction functions can be realized as presented in Section 3.

3. Self-Healing and Self-Reproduction.

Self-Reproduction is a challenging topic in Robotics. Based on the above SRS Architecture, the self-reproduction may come true, which is especially significant for those missions lasting long-term and experiencing a high risk of being destroyed without the possibility of human repair/replacement.

Another important function in the SRS is Self-healing, which automatically detects the errors and recovers from failures. Self-healing not only can reduce the risk of losing certain robots due to failures but also can salvage those robots that may have malfunctions caused by internal errors or external events such as collisions. The following subsections will introduce self-healing first and then discuss self-reproduction.

.

3.1. Self-Healing.

Accurate self-healing needs to involve more complicated computation, analysis and decision processes. Such analysis needs system-level models and makes decisions in a holistic manner. Suspect events lead to the analysis, including the abnormal cases that are detected by the virtual neurons. The self-healing needs to implement two important functions: autonomous diagnosis and autonomous curing.

Consequence Oriented Diagnosis and Curing.

The traditional diagnosis and curing mechanisms need to test and locate the bugs that are to be removed from the software codes. However, this procedure has to stop the program and recompile it. Thus, this traditional diagnosis and curing method is not suitable for the self-healing because in the real-time task running, the computers are not expected to reboot or stop the programs for recompilation. Moreover, the bugs and locations of the problems can be various, so it is impractical to ask the computer itself to precisely locate the errors and remove them automatically from the codes.

To make the self-healing more applicable, we hereby propose the idea of *Prescriptions* that have to be prepared beforehand and should be generic enough to heal most problems. Besides the generality, the prescription ought to recover the patient machines with minimum influence on the current services or users. Patterson *et al.* [10]

proposed to automatically reboot a machine if failures/problems are detected. This actually affects real-time service too much, especially during critical phases of mission operations. For example, in NASA's ANTS for exploring the asteroid belt, a reboot that interrupts a picocraft critical maneuvering operation could endanger the picocraft—possibly even resulting in a collision between two picocraft or concatenated collisions, causing the loss of multiple picocrafts.

Thus, we here propose that autonomous diagnosis and curing should be consequence-oriented. Even though there are numerous and various types of bugs, the consequence on the host robot performance may be similar. For example, the memory leakage may be caused by forgetting to release memory reserved for some objects when deleted. As a result, after a long-time run, the server of a robot fails to work properly due to the exhaustion of its memory. Such error may exist at different locations in various programs. However, no matter where the error exists or what types of the programs are, the consequence on the server is similar, i.e., consuming the memory. Therefore, the consequence-oriented self-healing is designed to recover the hosts by reclaiming the leaked memory without stopping the current processes or rebooting the computer. This can be understood as analogous to a Cleaner clearing the trash thrown by different persons instead of educating all of them not to throw the trash.

The consequence-oriented self-healing can achieve the requirements and solve those challenges that are the main obstacles to self-healing:

- Not to stop the program nor recompile it;
- Not to reboot the host to resolve small problems, thereby minimizing the effect on the services and users;
- Maintain the health of the hosts without downtime, which is very important in real-time applications or safety-critical systems;
- Overcome the heterogeneous challenges of various errors caused by different programs;
- There is no need to precisely locate the bugs in the code, which is not realistic in real-time self-healing for large-scale systems containing kaleidoscopic bugs;

The implementation of healing methods can be simplified and made generic by aiming at the consequence.

Based on the concept of consequence-oriented self-healing, the two important functions of autonomous diagnosis and autonomous curing can be implemented as follows:

Autonomous Diagnosis and Autonomous Curing.

The steps of autonomous diagnosis are depicted below:

1) If unexpected problems occur and persist, the virtual neuron reports them to the head neuron through the communicator.
2) The head neuron acts as the higher-level controller to coordinate the diagnosis process. It puts more effort in observing the suspicious patient-components by assigning more virtual neurons to help monitor more information and by increasing the report frequency.
3) Subsequently, the symptoms extracted from the data are analyzed together with the Cyber-Diseases Library. A hybrid diagnosis model will be implemented here to identify the possible diseases that cause the symptoms. The idea for the hybrid diagnosis is to narrow down the domain of possible candidate diseases. For example, given a set of symptoms observed by virtual neurons, the Fuzzy Logic will be first implemented on the data to narrow down the possible causes. Then, the domain can be further narrowed down using the Bayesian Network or Hidden Markov Model.
4) Then, one or several candidate diseases are identified with a probability assigned on each. According to the descending order of the probabilities, the autonomous curing process continues.

The Cyber-Disease Library is a knowledge-base that was originally built up based on the generic statistics, trained data and historical data. The complicated computation and analysis can be shared by the virtual neurons coordinated under the head neuron. The head also makes decisions at the same time; for example, if this component fails, who will take over its job? Also, the important data of this patient should be backed up on other peers before it fails totally.

After identifying a possible disease, the curing process starts to treat it. The autonomous curing process can be generally summarized as follows:

1) After the diagnosis, a prescription is determined for a corresponding disease and is sent to the virtual neuron located on the patient via the communicator.
2) The reflexor of the virtual neuron starts healing the patient-robot according to the steps/conditions of the prescription.
3) However, if the patient-robot itself is almost down with a poor performance, a neighbor robot is selected to remotely cure the patient based on the prescription via communicators.
4) In the worst case, the patient-robot is totally down or the communicator fails to work. A neighbor robot then will be assigned to physically connect to that patient-robot in order not to abandon it. There is a socket designed for physical connection. After the connection, the good robot can copy everything out of this patient-robot and then try to cure it via the prescription.
5) If the problem is identified and solved, it is reported to head neuron and then the two robots are separated

to continue their own jobs. Otherwise, if the patient cannot be cured and dies, the good robot that connects to the "corpse" checks the "organs" and throws away those useless or failed organs (to decrease weight) while retaining useful "organs" to enhance the computing/storage ability or power for the good robot. Those organs can also become a source for self-reproduction.

6) As in swarm robotics systems, failure in one robot is possible to occur in another, i.e. the common mode failure. Therefore, after the virtual neuron cures the robot, the corresponding failure and cure procedure should be broadcasted within the swarm to inform other neurons to help checking/debugging the related faults. This would be helpful in preventing the similar failure in other robots.

Thus, the SRS is endowing the ability for self-healing in an autonomic manner.

3.2. Self-Reproduction.

Self-Reproduction is evidently beneficial for the Swarm Robotic System, but the robotic reproduction is still a challenging topic. The major challenges include:

1) How do the parents generate the "components" for the next generation?
2) When is the mature time to start the self-reproduction and under which conditions?
3) How do the parents educate the kids to work for the mission?

The components of each robot should be designed as reassembly, pluggable and stretchable. The parent robots can offer the components for the next generation. There are three sources of the components:

1) The organs reclaimed from any dead robots. The mating process is coordinated by the head neurons to know whose redundant organs are matched so that they can be composed into another new robot.

2) The backup parts in a robot. Usually, each robot has certain important parts with backups through redundancies to increase reliability. This is the second source.

3) Dividable body. Those parts that can be divided into constituent components are the third source. For example, the framework can be broken in two and then each bone is stretchable to form two identical ones as the original one in the shape, like the stretchable antenna; or the framework is disassemble and reassemble, e.g. when a strut in a framework is composed of multiple substruts that can be unattached from each other and then used separately or reassembled in other combinations.

Mating is scheduled from the head-neurons acting as the higher-level controller. When to start the mating and reproducing is determined by two conditions. First, whether the system needs to add new members (such as the swarm of workers is not sufficient to finish a mission); second, whether the resources are enough to give birth to another new member. The first can be decided by the module of system models according to the evaluation of system reliability (the probability to finish certain tasks), the requirements of swarm intelligence, and the area that the system wants to cover for a mission. The second should also be determined by the communication among head neurons to know the organs attached on different robots, and then act as the mediator to induce them to marry.

When all the sources and timing are prepared, the mating process can start to make the next generations as the following steps:

1) The new framework is assembled from appropriate parts, either from spares possessed by the parent(s) or from a "bone yard" or repository stocked partly from components salvaged from dead robots.
2) After the framework is built up, the components are detached from the parents and then the parents plug them into the corresponding sockets. Magnetic force can help the plugging process.
3) After the hardware is prepared, copy the corresponding software into the new kid and then educate the kid by also inputting the parents' experience into the kid's knowledge base.

4. Implementation and Case Study.

The basic unit of the above prototype model is the virtual neuron which has been implemented in our TEGO research center, and the basic modules are developed and will be further improved. Some case studies based on those in-progress modules are conducted here to show the effectiveness and efficiency of self-healing on the servers. The system with and without the self-healing mechanism is compared as well.

4.1. Design of a case study for Memory Leakage.

The overall architecture contains different modules (as Fig. 1) that are controlled by the main system. The Virtual Neuron runs in the background and monitors the memory consumption, virtual memory allocations, CPU usage, network flow, service performance of the system and different processes.

In this case study, we compose a test program as the patient-process where the memory leakage is the problem. The moment system memory consumption ratio (perceived by the virtual neurons) crosses the threshold preset by the system model; the self-healing mechanism calls the module that can detect the specific abnormal process. The reflexor of Virtual Neuron then calls the suspendProcess subsystem, which freezes the process by suspending it. When the process is suspended, the resources allocated to that process are abstained and the system is brought to a safe state. Then the problem is reported to the higher-level controller for further diagnosis, and a suitable prescription is generated from the Cyber-Disease Library. Then, the actuator will run the prescription to solve the problem. In this case study, we let the machine itself operate as the actuator. In practice, it is not difficult to extend to other peers as actuators by applying network programming.

If the diagnosis module finds that the reported problem is benign, the suspended process is resumed. However, if the reported problem is analyzed as a malicious or dangerous one, then the reflexor of the virtual neuron will operate the first prescription (Reclaim Leaked Memory) assigned from the Cyber-Disease Library. If the first prescription succeeds, then the disease is healed. Otherwise if the first prescription fails, the second prescription (kill the process) will be sent to the actuator, then the problem can be solved at the second-level, though it is not so intelligent. If the problem still persists, the third prescription (Reboot the machine) is sent to the actuator to record the states, reboot the machine, restart all processes, and return to the recorded states to continue the job.

Some context information is monitored and compared in the following cases. The definitions of these measures:
System Available Memory: The amount of free memory of the computer system that is available to be allocated.
Virtual Memory Pages Allocation (VMPG): The number of pages of the virtual memory allocated to a certain process.
System Page Faults/sec: The number of pages faulted per second, which includes both hard faults (those that require disk access) and soft faults (where the faulted page is found elsewhere in physical memory.)

4.2. Comparisons on Some Measures.

Under the above design and measures, the implementation is made on the machine that emulates the brain of a robot. A test program is composed with memory leakage, and starts running on the 30th second. The amount of available memory in the system by using our self-healing mechanism is depicted by Fig. 2. As observed, the virtual neuron suspected the problem after 66 seconds, and then suspended the malicious process for diagnosis. However, the diagnosis module made the first decision, it is tolerable and maybe benign, so the process resumed at 96 seconds, but a prescription has been sent to the local actuator, i.e., "if the available system memory is less than 100M, the memory that has been leaked by the process is reclaimed". Hence, at 144th seconds, the actuator starts gleaning the leaked memory and recovers the system back to the original good performance as a brand new one.

Then, when we do the same experiment without equipping the self-healing function, the server is totally down after 128 seconds when there is no available memory in the system, as depicted by Fig. 3.

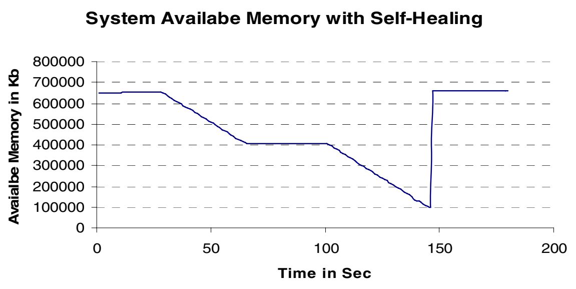

Fig. 2. Available memory in a system with self-healing.

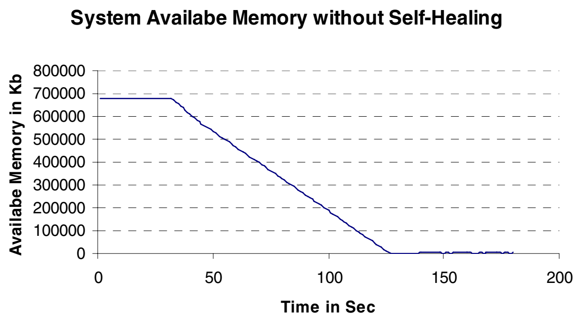

Fig. 3. Available mem in a system without Self-healing.

System Page faults per sec (*PFPS*) is analyzed as another context measure. When the process is leaking the memory, the *PFPS* keep increasing that is malicious. When the overall memory of the system reaches the threshold values, the process is suspended by the Virtual Neuron and the *PFPS* drops to a small value, which indicates good performance with few disk accesses. When the bad process is resumed after the first diagnosis (benign), the *PFPS* continue to increase. Finally, when the received prescription is executed, the *PFPS* drop back to the normal low value. The whole process is depicted by Fig. 4: at the beginning, the *PFPS* is low; then, when the memory leaked for a while, the *PFPS* is increased; at the 50th second the process is suspended and *PFPS* drops to near zero value; and when resumed at the 80th second, the *page fault /sec* value shoots up, and finally, when the prescription is exercised by the actuator at the 115th second, the system is recovered.

Consider the system without self-healing. The *PFPS* increases and maintains at a high value, which is depicted in the Fig. 5.

In addition, the reason for suspending the suspect process at the abnormal stage is to decrease the possibility of harming the good process, which might temporarily look like an abnormal one. When the process is suspended, the diagnosis starts working to investigate the process, which can be resumed when it has found a solution (prescription) for it.

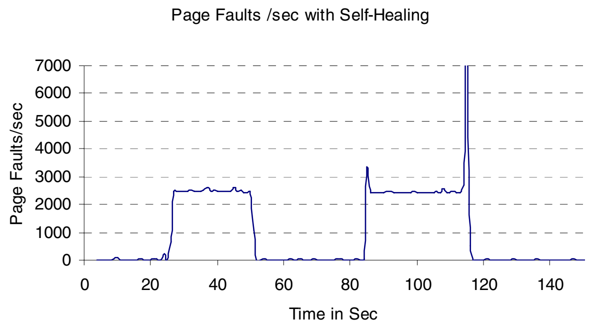

Fig. 4. Page Faults with the presence of self-healing.

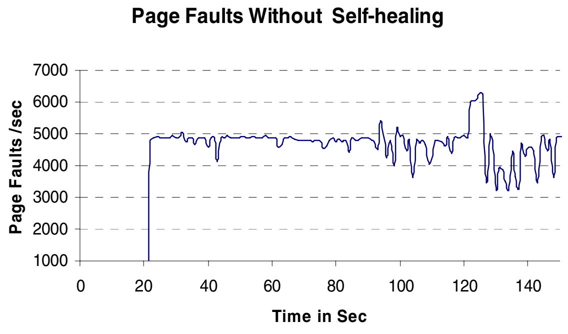

Fig. 5. Page faults per second without Self-healing.

4.3. Influence of bad process on good process.

In this section, we see the effect of the bad process on a good process that may be critical in maintaining the basic functions of the robot. The performance of each process is collected. The virtual neuron gets the context measure of *Virtual Memory Pages Allocation* (defined in 4.1) to show this influence.

Initially, a good process and a bad process (with memory leakage) are both started. Even though the bad process did not take the whole system down, we find that the bad process obviously affects the performance of the good one. It is observed that the system resources are occupied due to the bad process's memory leakage. For example, the *Virtual Memory Pages Allocation* (VMPA) represents how much resource of the virtual memory is allocated on a certain process. This case watches the allocated virtual memory on the good process under the effect of the bad process running. It is observed that VMPA of the good process drops down, almost to zero, indicating that the process of virtual memory allocation is minimal. When the memory consumption of the bad process crosses the threshold, it is detected by the virtual neuron as a suspect process and is thereby suspended. After suspending the bad process, the good one functions normally and this is shown by its VMPA, which goes back to normal. Then, the self-healing system does the investigation to diagnose the bad process. The prescription is received and the bad process is resumed to continue affecting the good process as the VMPA of the good process drops down considerably. After the prescription is executed to reclaim the leaked memory, the good process comes back to its normal state. The whole process is depicted by Fig. 6.

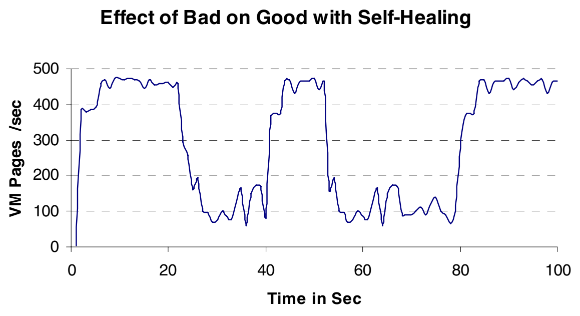

Fig. 6. Bad process affects good one with self-healing.

Then, the same experiment, without activating the self-healing system, was conducted. The good process is started as well as the bad process. Initially, there is not much variation in the VMPA, with about 500 pages of virtual memory allocated to the good process every second. As time goes by, the bad program, due to the memory leak, affects the VMPA of the good one. This influence can be observed as the VMPA reduces considerably to a small value. This phenomenon persists and the bad process totally blocks the resources for the good one. The whole process is depicted by Fig. 7.

Without the self-healing system, it is apparent that the problems of any single bad process can definitely affect other processes (downgrade till failure), so the failures are able to propagate if they cannot be quickly detected and correctly healed. From the comparisons in the above cases, the novel self-healing system is validated very effective and useful.

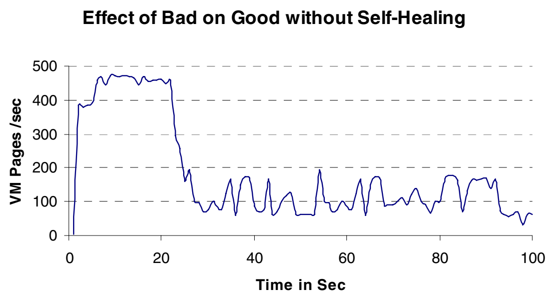

Fig .7. Bad process on good one without self-healing.

5. Conclusion

This paper presented a self-healing and self-reproduction mechanism based on the virtual neurons for the SRSs. The prototype architecture of the virtual neuron is designed as other autonomous diagnosis and curing modules. Based on this prototype model, the SRS possesses the self-healing and self-reproduction characteristics that were challenging topics before. The consequence-oriented prescription mechanism was presented, which is more suitable for autonomic computing in the real-time robotics system. The virtual neurons and self-healing function had been implemented in the computer systems at the TEGO research center. Several cases were studied to show the effectiveness of the self-healing function, and were analyzed to show the effects of the novel mechanism on the bad and good processes. The results demonstrated that the self-healing mechanism makes the system much more reliable and the performance is much improved, not only against failures, but also against failure propagations.

However, this interesting research topic in autonomic computing for self-protection is still in its infancy and more research needs to be done in the near future. Though the novel technologies have been validated in our TEGO center, some more real implementations and practical applications are required. This prototype system can also be applied in some safety-critical missions that require high reliability and high autonomy, such as NASA's ANTS-like projects, as part of possible future research.

References

[1] Y.S. Dai, M. Xie, K. L. Poh, "Markov renewal models for correlated software failures of multiple types" *IEEE Transactions on Reliability*, 2005 vol. 54, no. 1, pp. 100-106.

[2] Y.S. Dai, T. Marshall, X. H .Guan, "Autonomic and dependable computing: moving towards a model-driven approach", *Journal of Computer Science*, Accepted for publication 2006.

[3] K. Fukuda, D. Funato, K. Sekiyama, F. Arai, "Evaluation on flexibility of swarm intelligence system", *Proceedings of the IEEE International Conference on Robotics & Automation*, , vol. 4, 1998, pp 3210-3215.

[4] A.T. Haytes, A. Martinoli, R M. Goodman, "Swarm robotic odor localization", *In Proceedings of the 2001 IEE/RSJ International Conference on Intelligent Robot and Systems*, 2001, pp. 1073-1078.

[5] J. Kephart, D. Chess, "The vision of autonomic computing", *Computer*, 2003, vol. 36, no. 1, pp. 41-50.

[6] D. Kewley, J. Bouchard, "DARPA information assurance program dynamic defense experiment summary", *IEEE Transactions on Systems, Man, and Cybernetics – Part A*, 2001, vol. 31, no. 4, pp. 331-336.

[7] O. Kreidl, T. Frazier, "Feedback control applied to survivability: a host-based autonomic defense system", *IEEE Transactions on Reliability*, 2004, vol. 9, no. 2, pp. 148-166.

[8] C R. Kube, C R. Bonabeau, "Cooperative transport by ants and robots, *Robotics and Autonomous Systems"*, 2000, vol. 30, pp. 85-101.

[9] P. Motuzenko, "Adaptive domain model: dealing with multiple attributes of self-managing distributed object systems", *Proceedings of the 1st International Symposium on Information and Communication Technologies*, 2003, pp. 549-554.

[10] D. Patterson, A. Brown, P. Broadwell, et al., "Recovery oriented computing (roc): motivation, definition, techniques, and case studies", *Technical Report CSD-02-1175, Univ of California-Berkeley*, 2002, pp. 1-25.

[11] L. Paulson, "Computer system, heal thyself", *Computer*, vol. 35, no. 8, 2002, pp. 20-22.

[12] G. Prencipe, "CORDA: distributed coordination of a set of autonomous mobile robots", *In Proceedings 4th European Research Seminar on Advances in Distributed Systems*, 2001, pp. 185-190.

[13] R.K. Sahoo, I. Rish, A.J. Oliner, , et al. , "Critical event prediction for proactive management in large-scale computer clusters" , *the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003, pp. 426-435.

[14] W. Truszkowski, M. Hinchey, J. Rash, C. Rouff, "NASA's swarm missions: the challenge of building autonomous software", *IT Professional*, vol. 6, no. 5, 2004, pp. 47-52.

[15] Xie, M., Dai, Y.S., Poh, K. L. *Computing Systems Reliability: Models and Analysis*, Kluwer Academic Publishers: New York, NY, U.S.A, 2004.

[16] Zou, X., Ramamurthy, B., Magliveras, S., *Secure Group Communication over Data Networks*, Springer, ISBN: 0-387-22970-1. 2004.

[17 J. Pena, , M.G. Hinchey, R. Sterritt, "Towards Modeling, Specifying and Deploying Policies in Autonomous and Autonomic Systems Using an AOSE Methodology", *ease, pp. 37-46, Third IEEE International Workshop on Engineering of Autonomic & Autonomous Systems (EASE'06),* 2006.

[18] W. F. Truszkowski, M. G. Hinchey, J. L. Rash, C. A. Rouff, "Autonomous and Autonomic Systems: A Paradigm for Future Space Exploration Missions", *Systems, Man and Cybernetics, Part C, IEEE Transactions on, vol. 36, no. 3* 2006.